



ENGINEERING EMERGENCE

APPLIED THEORY
FOR
GAME DESIGN

JORIS DORMANS

Engineering Emergence

Applied Theory for Game Design

Joris Dormans

Joris Dormans, 2012.
ISBN: 978-94-6190-752-3



This work is licensed under the Creative Commons Attribution-NonCommercial 3.0 Netherlands License. Visit <http://creativecommons.org/licenses/by-nc/3.0/nl/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Engineering Emergence

Applied Theory for Game Design

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor

aan de Universiteit van Amsterdam

op gezag van de Rector Magnificus

prof. dr. D.C. van den Boom

ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit

op vrijdag 13 januari 2012, te 11.00 uur

door

Joris Dormans

geboren te Geleen

Promotiecommissie:

Promotor:

prof. dr. ir. R.J.H. Scha

Co-promotor:

dr. ir. J.J. Brunekreef

Overige leden:

dr. ir. A.R. Bidarra

prof. dr. P. van Emde Boas

prof. dr. P. Klint

prof. dr. ir. B.J.A. Kröse

dr. M.J. Mateas

prof. dr. E.S.H. Tan

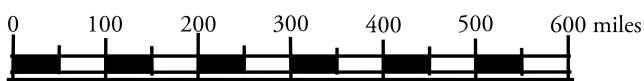
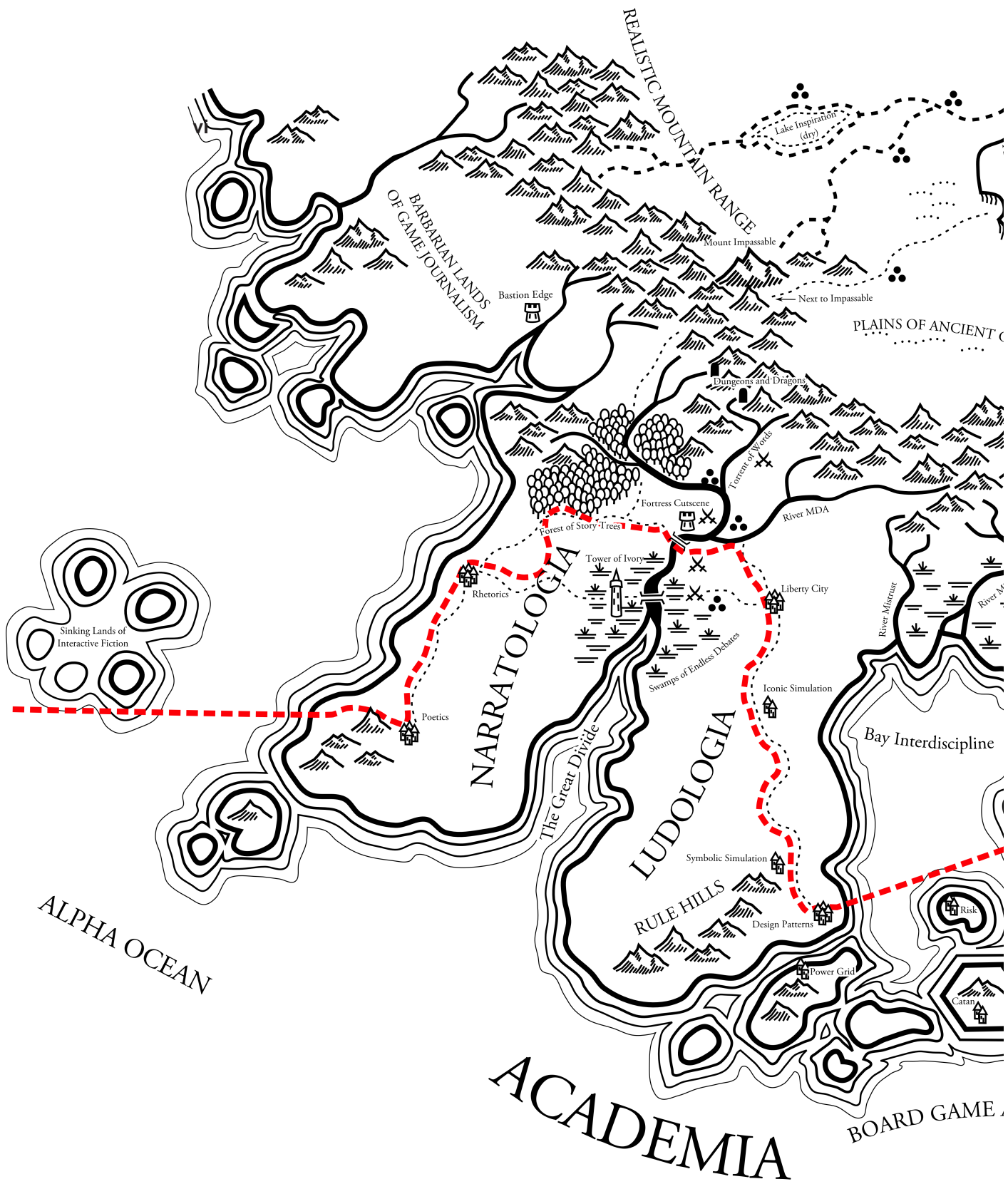
Faculteit der Geesteswetenschappen

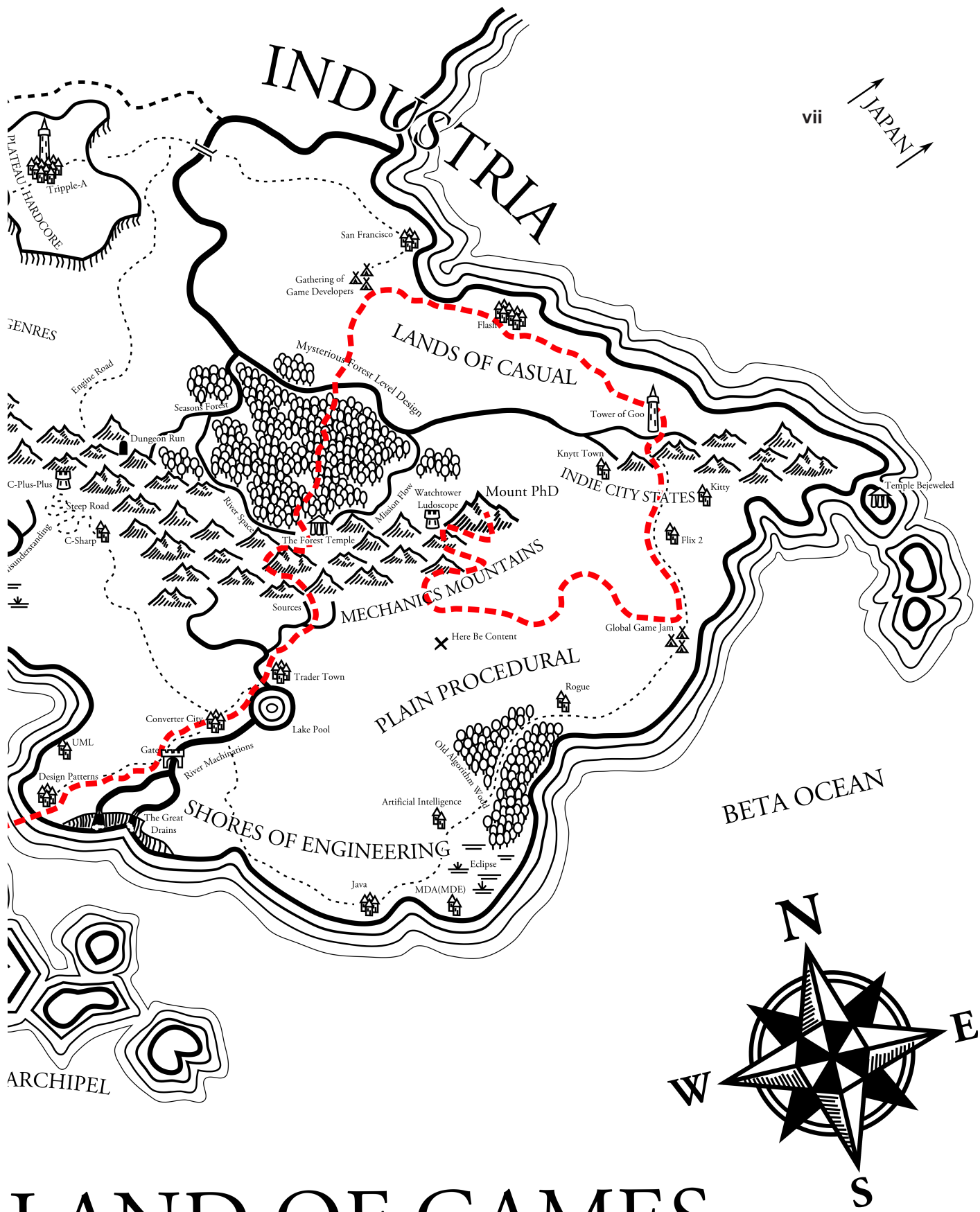
Contents

Map of the Land of Games	vi
Preface	ix
1 Introduction	1
1.1 Games	2
1.2 Mechanics	6
1.3 Game Classification	10
1.4 Emergence and Progression	13
1.5 Emergence	16
1.6 Progression	19
1.7 Approach and Dissertation Outline	21
1.8 Terminology	22
2 Rules, Representation and Realism	25
2.1 The Iconic Fallacy	27
2.2 Indexical Simulation	30
2.3 Symbolic Simulation	31
2.4 Less Is More	33
2.5 Designing Emergence	37
2.6 Conclusions	40
3 Game Design Theory	43
3.1 Design Documents	44
3.2 The MDA Framework	46
3.3 Play-Centric Design	48
3.4 Game Vocabularies	49
3.5 Design Patterns	51
3.6 Mapping Game States	53
3.7 Game Diagrams	57
3.8 Visualizing Game Economies	59
3.9 Industry Skepticism	61
3.10 Conclusions	64
4 Machinations	67
4.1 The Machinations Framework	68
4.2 Flow of Resources	71
4.3 Flow of Information	75
4.4 Controlling Resource Flow	78
4.5 Four Economic Functions	80
4.6 Feedback Structures in Games	83
4.7 Feedback Profiles	87
4.8 Feedback Analysis and Recurrent Patterns	90
4.9 Implementing Machinations Diagrams	93

4.10	Randomness and Nondeterministic Behavior	96
4.11	Case study: SIMWAR	100
4.12	Conclusions	104
5	Mission/Space	109
5.1	Level Layouts	110
5.2	Tasks and Challenges	113
5.3	The Mission/Space Framework	115
5.4	Mission Graphs	117
5.5	Space Graphs	121
5.6	Level Analysis: The Forest Temple	125
5.7	Mission-Space Morphology	129
5.8	A Software Tool for Level Design	130
5.9	Conclusions	133
6	Integrating Progression and Emergence	135
6.1	From Toys to Playgrounds	136
6.2	Progression through Structured Learning Curve	137
6.3	Economy Building Games	142
6.4	A Mismatch Between Mission and Space	146
6.5	Mechanics to Control Progression	149
6.6	Feedback Mechanisms for Locks and Keys	152
6.7	Progress as a Resource	154
6.8	Conclusions	159
7	Generating Games	161
7.1	Game Design as Model Transformation	162
7.2	Formal Grammars	164
7.3	Rewrite Systems	166
7.4	Graph Grammars	168
7.5	Shape Grammars	170
7.6	Example Transformation: Locks and Keys	173
7.7	Generating Space	174
7.8	Generating Mechanics	180
7.9	Procedural Content in Games	185
7.10	Adaptable Games	187
7.11	Automated Design Tools	191
7.12	Conclusions	193
8	Conclusions and Validation	197
8.1	Structural Qualities of Games	198
8.2	Validation	200
8.3	Teaching Game Design	201
8.4	Building Prototypes	205
8.5	Academic and Industry Reception	208
8.6	Omissions	209
8.7	Future Research	210

Contents	v
8.8 Final Conclusions	211
A Machinations Overview	215
B Machinations Design Patterns	217
B.1 Static Engine	217
B.2 Dynamic Engine	220
B.3 Converter Engine	223
B.4 Engine Building	227
B.5 Static Friction	229
B.6 Dynamic Friction	231
B.7 Attrition	234
B.8 Stopping Mechanism	238
B.9 Multiple Feedback	240
B.10 Trade	242
B.11 Escalating Complications	244
B.12 Escalating Complexity	246
B.13 Playing Style Reinforcement	248
C A Recipe for Zelda-esque Levels	253
Summary	259
Nederlandse Samenvatting	263
Index	267
Bibliography	273
Ludography	285





LAND OF GAMES

Preface

My road to the summit of Mount PhD located in the heart of the Land of Games has been a long and wondrous journey. During its many detours I met many great and inspiring people. Some I met briefly, others accompanied me for a while, none failed to brighten up my journey with valuable advice and unwavering support. Looking back at the curvy route, I now know that all twists and turns were necessary to bring me to the place I am today.

One might say that I am a native of the Land of Games. During my childhood I spend a great many hours playing games of all sorts: board games, pen-and-paper roleplaying games, and video games. And we invented our own from the very start. Countless are the sheets of paper I used to design boards for overly ambitious games, dungeons for endless roleplaying sessions, and plans for games that I never quite finished programming on our Commodore 64.

Growing into adulthood and becoming a student, I left the land of games on what would prove to be the longest detour of all. Years I spent in the fine halls of education on the far side of the Alpha Ocean. During this period I gained knowledge and skills that would prove to be trusty tools on the journey that finally led me here.

Returning to these shores was a joy. I had grown wiser, and my vision had been sharpened. Some things had changed, many others had remained the same. Immediately I knew where I wanted to go. Finding the right road did not always prove easy, but I can now say I succeeded. I must thank many people for the support, advice and company along this journey.

Claartje, my mother, showed me at an early age that through strength of will and firm determination you can shape your destiny and climb any mountain you will find on your path. Mom, no matter what happens, your inner strength will be my guiding light forever.

Marije has been my loving companion since long before I set out on this path and has been with me all the way. Now you stand beside me at this journey's conclusion as my paranimf. Without your moral support and textual advice I would never have reached this high.

My promotor, Remko fulfilled the role of mentor with excellence. Despite being a relative stranger to the Land of Games, you never failed to ask all the right questions when I needed to reset my course.

My co-promotor, Jacob, provided with me with the right map to successfully navigate the Shores of Engineering. Without that map, I would never have been able to travel as far and wide as I did.

I also thank the other members of the committee: Ben, Ed, Michael, Paul, Peter and Rafael. Some of you I met in distant lands, many of you I met along the way. I am very grateful that you took the time to accompany me during the final ascent.

I could not wish for a merrier band of travel companions than my colleagues at the Hogeschool van Amsterdam. We shared many adventurous days on the winding paths that led me to this mountain. Your trust in the course I set and support in paving the roads so that others might easily follow must not go

unmentioned.

My students never failed to surprise me, especially when they were replicating parts of my journey. They always picked their own route, which for me was a source of inspiration.

I must also thank my family and friends whose interest in my progress was both sincere and supportive. A deep bow to all of you who were once my opponents in so many well-played games.

From all people along my journey I will thank two more persons. Firstly, Carla who introduced me to all the right people which finally allowed me make landfall on these shores after being lost at sea for a while. And finally, last but not least, Jasper my other paranimf whose friendship goes back all the way back to our childhood. The many games we enjoyed kept the memory of these lands alive and paved the way for my return.

There is no single sentence describing what makes games attractive.

Jesper Juul (2003)



Introduction

Designing games is hard. Although games have been around for a very long time, it was the rise of the computer game industry over the past few decades that caused this problem to become prevalent. During its short history the computer game industry has grown from individual developers and small teams towards multi-million dollar projects involving hundreds of employees. In the contemporary game industry there is little room for mistakes: the financial stakes have grown too high. Today, more than any time before, there is a need for a better understanding of the process of designing a successful game in order to prevent such mistakes; there is a need for better applied theory and intellectual tools to aid game designers in their task.

At the same time, more people are playing video games than ever before. A wider audience means that there is an ever increasing demand for games with quality gameplay. As game players get more experienced they grow an appetite for ever more sophisticated games. Compared to other forms of art and media, computer games are a fairly recent invention. There is still plenty of room for development and innovation.

The general premise of this dissertation is that the difficulties in designing games lie within the nature of games as complex rule based systems that exhibit many emergent properties on the one hand, but must deliver a well-designed, natural flowing user experience on the other. In facing these difficulties, the game designer's tool box is quite empty. The nature and emergent behavior of games is poorly understood. Level design has been one way to harness a game's emergent behavior, by restricting the gameplay to a series of relatively simple tasks loosely strung together by a storyline. However, high-quality content is expensive to produce. Games with many hand-crafted levels are expensive to produce, and fail to exploit the true expressive power of open game worlds that emerge from rule systems.

This dissertation examines the nature of emergence in games in order to construct applied theory to deal with emergence in games head-on. This theory will enable the designer to get more grip on the elusive process of building quality games displaying emergent behavior. The theory developed in this dissertation applies to game mechanics and levels. However, where many scholars and designers treat levels and mechanics as two vastly different elements of game design,

this dissertation attempts to integrate the two: for both rules and levels, this dissertation seeks to find formal, abstract representations through which the process of designing these aspects of games might be elevated and unified. Through these representations the material that game designers work with should become more tangible. This leads to the central research question of this dissertation: *what structural qualities of game rules and game levels can be used in the creation of applied theory and game design tools to assist the design of emergent gameplay?*

The development of software tools to assist game designers in their task is an important aspect of this dissertation. All too often, design theory has been created in isolation from the practice of game design. The creation of software tools that implement the theories presented here, means that these theories have to be very concrete and applicable. In addition, it allows the automation of certain parts of the design process. By automating these parts, designers will be assisted in their work and can focus on those aspects of design that require the most of human creativity and ingenuity.

This chapter introduces the central notions of this dissertation: games, gameplay, mechanics, levels, emergence, and progression. It also outlines the general approach and the contents of the chapters that follow.

1.1 Games

What are games? Many people play them, but only a few stop to contemplate their nature. The study of games, especially the study of games in its current form, is very young. It was only in 2001 that Espen Aarseth declared that year to be “year number one” of game studies (Aarseth, 2001). That year saw the launch of the first peer-reviewed online journal and the first international academic conference dedicated to games. Games were studied before, but it was not until 2001 that game studies gained enough momentum to be recognized as a separate academic discipline.

The study of games has been a multi-disciplinary affair from the beginning, with researchers from different fields studying games from different perspectives. The first few years of game studies were characterized by a fierce debate between narratologists and ludologists. The former group comprised academics, often with a background in literature, who had been studying games from that perspective for a while. They regarded games as a new medium for storytelling and placed games in the context of literature and media studies (Laurel, 1986; Murray, 1997; Ryan, 2001). The ludologists opposed this position, for them games are rule-driven play experiences first and foremost. The story and visuals are secondary to rules which are the most critical factor for game quality. Their argument is that good rules with visuals and story of lesser quality still make for a good game, whereas the opposite is not true (Eskelinen, 2001; Juul, 2005).

Today both positions are considered rather extreme. It is difficult to find somebody who would maintain that the paradigms used to study stories in literature or cinema apply directly to games. You cannot ignore rules, interactivity

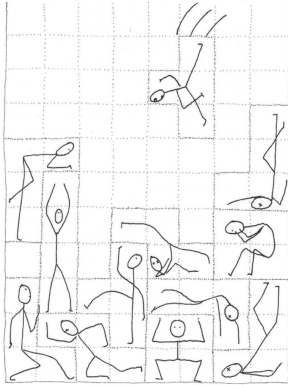


Figure 1.1: Raph Koster’s hypothetical Mass Murderers Game based on TETRIS.

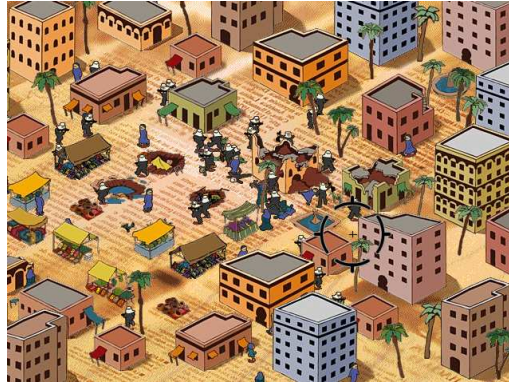


Figure 1.2: Shooting missiles at terrorists in the satirical SEPTEMBER 12.

and gameplay in any study of games. On the other hand, the reskinning of TETRIS into the hypothetical “Mass Murderer Game” by Raph Koster (see figure 1.1) where the player tries to fill pits with awkwardly shaped dead bodies, clearly illustrates that story and visuals do affect the experience of play (Koster, 2005b, 166-169). The biting irony of a game like SEPTEMBER 12 where the player is invited to shoot missiles at terrorists in an Arabic city and to explore the consequences of that action is only made possible by the sharp contrast between rules that support simple, typical gamelike shooting action and the game’s meaningful reference to a very real situation outside the game (see figure 1.2). Games do not exist in isolation but are part of a heterogeneous media landscape and the social structures from which stories derive their meaning. In this case it is worth noting that SEPTEMBER 12’s developer, Gonzalo Frasca, is also a prominent ludologist and was in fact the first game researcher that coined the term ludology (Frasca, 1999).

The examples of the MASS MURDERER GAME and SEPTEMBER 12 also illustrate nicely that in games, what matters most is what the player does. These actions are determined by rules on the one hand, but on the other a game’s art and story frame these actions and give them meaning. In a game, rules set up possible interactions, but through clever design of levels developers have some control over the order in which players encounter game elements and the challenges they pose. It is through levels that developers primarily control the sequence of actions.

The general consensus in game studies is that games are rule based artifacts designed to be experienced by one or more players, in which they strive to achieve some sort of goal. Without rules there would be no game, but the structured experience of the player is not unlike the structures and experience encountered in other media.

After close examination of eight different definitions of games, from histo-

rian Johan Huizinga to game designer Greg Costikyan, Katie Salen and Eric Zimmerman define games as follows:

“A game is a system in which players engage in artificial conflict, defined by rules, that results in a quantifiable outcome.” (2004, 80)

In their definition *system*, *players*, *artificiality*, *conflict*, *rules* and *quantifiable outcome* are the key notions. All games are *systems* consisting of many parts that form a complex whole (Salen & Zimmerman, 2004, 55). The *system* is defined by *rules* that determine what *players* can and cannot do. Following those rules, players engage in *conflict* against each other or against the game system. The conflict is *artificial* in the sense that the game is set apart from real life in both time and space, a space where the players submit to the rules of the game. In this sense, games are often said to take place within a “magic circle”, after the work of Johan Huizinga (1997). Finally, a game has a *quantifiable outcome*: players can win or lose, or measure their performance with some sort of score.

Salen and Zimmerman’s definition resembles many other definitions, even those not investigated by themselves. Mark J. P. Wolf uses the elements *conflict*, *rules*, *player ability* and *valued outcome* to define games (2001, 14). Alexander Galloway states that: “a game is an activity defined by rules in which players try to reach some sort of goal” (2006, 1). Ernest Adams and Andrew Rollings identify *rules*, *play*, *goals* and *pretending* as key elements of games. The latter element is linked to the magic circle and by extension to Salen and Zimmerman’s notion of *artificiality* (Adams & Rollings, 2007, 5-11). For Tracy Fullerton a game is “a closed, formal system that engages players in structured conflict and resolves its uncertainty in an unequal outcome” (2008, 43).

Jesper Juul examines many of the same definitions of games, including the definition of Salen and Zimmerman. He concludes that the following six features define games (Juul, 2005, 36):

1. Games are *rule based*,
2. and have *variable, quantifiable outcomes*,
3. which are affected by the *effort* of the player,
4. and which are assigned different *values*,
5. and to which the player is *emotionally attached*,
6. and which *consequences are negotiable*.

Of these six features only the first three are properties of the game as a formal system. The other three are either properties of the relation between the game and the player or the relation between the game and the rest of the world (Juul, 2005, 37).

Compared to Salen and Zimmerman, Juul’s definition incorporates a few extra elements. First, in Juul’s definition the *outcomes* of a game are not only

quantifiable, they are also variable. Games must have different outcomes to work as game. As soon as a game will always have the same outcome, there is little point in playing. This can happen when two players of vastly different levels are competing. When one of them is sure to win, when the outcome is known before the start, the game will cease to function as a game. Second, for Juul the player must be able to affect a game by putting in an *effort*. Without effort, the player's actions are meaningless and the player will never become *emotionally attached* to the outcome of the game. For Juul this makes all games of pure chance, where players cannot affect the outcome in any way, borderline cases (Juul, 2005, 44). Thirdly, Juul pays more attention to the relation between the game and the player, and to the relation between the game and the world as is indicated by his last four points.

At the same time, Juul leaves out the *artificiality* of Salen and Zimmerman's definition. In Salen and Zimmerman's definition *artificiality* plays a similar role to Johan Huizinga's concept of the 'magic circle' through which games create a reality outside real life (Huizinga, 1997). Although the magic circle is arguably porous (Copier, 2007) the artificial nature of games is without question. Game rules are made by designers and upheld by players to create an experience; players submit to these rules to experience the game. Games are also constrained by 'rules' that exist prior to the game, such as the law of gravity which constrains almost any sport (Juul, 2005, 58), but all games add rules to set up artificial goals, conflict and challenges. The game creates a space where the game is played, whether or not that space has clear boundaries.

For this dissertation I choose to build on Salen and Zimmerman's definition of games, although I do add Juul's *player effort and ability* to affect the *variable outcome* of the game. I choose to disregard Juul's other additions as this dissertation focuses on games as formal systems and not on the relation between games and players. Thus, for this dissertation, games are defined as follows:

A game is a system in which *players* engage in *artificial conflict*, defined by *rules*, that results in a *variable, quantifiable outcome* affected by *player effort and ability*.

Gameplay, a key notion associated with the players actions' and experience of play, is more difficult to define. Gameplay somehow consists of what the player *does*. At the same time, the term is used to describe a quality possessed by games themselves. Reviewers of games often talk about gameplay in this sense. Used in this way, "gameplay has become synonym with good gameplay" as Niels 't Hooft once remarked.¹ That is to say, whether or not a game *has* gameplay, has become an assessment of its quality: good games have gameplay.

For this dissertation I will use gameplay in this sense. When designers are working to create gameplay, they are always working to create a compelling game experience. The game, as a product, is the prime source for this experience. What follows is that gameplay somehow emerges from the way a game is

¹Niels 't Hooft is a freelance game journalist who works for Basher.nl and the Dutch newspaper NRC Next. He made this statement during a GameLab meeting on gameplay on February 2, 2011 in Pakhuis de Zwijger, Amsterdam.

constructed. It is these structural qualities of games as rule based systems that are the focal point of this dissertation.

1.2 Mechanics

When the game design community talks about game systems, they prefer the term “game mechanics” over “game rules”. “Game mechanics” is often used as a synonym for rules but the term implies more accuracy and is usually closer to an implementation. Although implementation here is still relatively independent from any platform or medium. Game designers Ernest Adams and Andrew Rollings explain the difference between the two with the following example: the rules of a game might dictate that in a game caterpillars move faster than snails, but the mechanics make the difference explicit; the mechanics specify how fast caterpillars move and how fast snails move. Mechanics need to be accurate enough for game programmers to turn them into code without confusion or for board game players to execute them without failure; mechanics specify all the required details (Adams & Rollings, 2007, 43).

In a similar vein, Morgan McGuire and Odest Chadwicke Jenkins state that “Mechanics are the mathematical machines that give rise to gameplay; they create the abstract game” (2009, 19). With that they point out that mechanics are media-independent: they are amongst those parts of games that are separable from images and sounds and might actually be transposed from one medium to another: a board game might be recreated as a computer game with different art and a different theme without altering the mechanics.

Game designers are perfectly comfortable talking about a “game mechanic” in the singular form (McGuire & Jenkins, 2009; Brathwaite, 2010). With this they are not referring to a person who is skilled in dealing with game mechanics, as the common use of the singular form “mechanic” would imply.² Instead, they are referring to a single game mechanism that governs a certain game element. One such mechanism might include several rules. For example, the ‘mechanic’ of a moving platform in a side-scrolling platform game might include the speed of the platform’s movement, the fact that creatures can stand on it, the fact that when they do they are moved along with it, but also the fact that the platform’s velocity is reversed when it bounces into other game elements, or perhaps after it has traveled a particular distance. In this dissertation I prefer to use “mechanism” as the singular form indicating a single set of game rules associated with a single game element or interaction.

Some mechanics may be more central to a game than others. The term “core mechanics” is often used to indicate those mechanics that the player interacts with most frequently and have the biggest impact on the gameplay (Adams & Rollings, 2007; McGuire & Jenkins, 2009). Moving and jumping, for example, are core mechanics of most platform games. In contrast, the mechanics that specify that players gain one extra life for every hundred stars they collect, might or might not be considered to be the core of a game. For a game where

²The Oxford Advanced Learner’s Dictionary lists “a worker skilled in handling or repairing machines” as the sole meaning of the word mechanic.

the extra life is just a nice bonus, it probably is not a core mechanism, but for a game where stars are abundant and players lose lives easily it probably is. The distinction between core mechanics and non-core mechanics is not clear-cut; even for the same game, interpretation of what is core and what is not can vary between designers or even between different moments within the game.

Mechanics have come to indicate many different types of rules in games. The term sometimes denotes mechanics in the physics sense: the science of motion and force. In games characters commonly move, jump or drive vehicles. Knowing where a game element is, in what direction it is moving and whether or not it is intersecting or colliding with other elements make up the bulk of all calculations in many games. Here mechanics might be interpreted quite literally as the implementation of the physical laws that govern motion and force within the game. At the same time, games also include mechanics that have nothing to do with physics: for example, mechanics that specify how many coins need to be collected to gain an extra life. The mechanics that deal with power-ups, collectibles and other types of game resources constitute something that might be called an internal economy (Adams & Rollings, 2007, 331-340). The nature of economic mechanisms and game physics is different in a number of crucial ways. One problem of using the term mechanics for both is that it obscures these crucial differences.

Physics in modern games tends to be simulated with accurate mechanics that create near continuous game simulations. A game object might be positioned half a pixel more to the left or right and this might have a huge effect on the result of a jump. In contrast, the rules of an internal economy tend to be discrete; game elements and actions are a finite set that do not allow any gradual transitions: in a game you usually cannot pick up half a power-up. This continuous nature of game physics versus the discrete nature of game economies has consequences for the medium (in)dependence of games, the nature of the player interaction, and even for the opportunities for design and innovation. These effects are discussed below.

Due to its continuous nature, the implementation of physics tends to be much more closely tied to the medium or platform than a game economy is. Economic mechanics are indeed separable from a game's medium, but physics not to the same extent. For example, a game that relies heavily on physics can not be easily mediated as a board game. Creating a board game for SUPER MARIO BROS. (see figure 1.3) where the gameplay originates from moving and jumping from platform to platform is very difficult. The continuous physics of a platform game translate poorly to the discrete nature of board games. A die only has so many sides, and to keep the game accessible overly complex calculations are best avoided. In platform games physical dexterity matters, just like a whole myriad of physical skills determine whether or not somebody is good at playing real-life football; those skills would be lost in a board game. SUPER MARIO BROS. is probably better mediated as a physical course testing players' real running and jumping abilities. The point is, a rule that states you can jump twice as high after picking up a certain item, can be easily translated between different media, whereas rules that implement the physics of a jump cannot. The physical



Figure 1.3: The gameplay in SUPER MARIO BROS. emerges in large part from continuous physics for running and jumping.

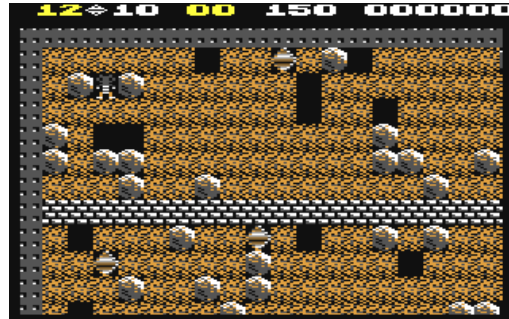


Figure 1.4: The physics in BOULDER DASH is implemented through discrete system. Objects such as diamonds and boulders are always aligned to the game's grid of tiles.

mechanics of a game seem to be bound more closely to the medium than the discrete rules that govern a game's economy.

Interestingly, when we look back at the early history of platform games and other early arcade games, physics were often handled quite differently, much more discrete, one might say. The moves in DONKEY KONG were much less continuous than they were in SUPER MARIO BROS.. In BOULDER DASH (see figure 1.4) gravity is simulated by moving boulders down at constant speed of one tile every frame. It might play slowly, but it is possible to create a board game for BOULDER DASH. In those days the rules that created the game's mechanics (in the physical sense) were not that different from other types of game rules. But times have changed. Today the physics in a platform game have grown so accurate and detailed that they have become impossible, or at least inconvenient, to represent with a board game.

With discrete rules it is possible to look ahead, to plan moves, create and execute complex strategies. Although this does not need to be easy, it is possible and players are encouraged to do so. Player interaction with this type of rules is much more on a strategic level. On the other hand, once players grasp the physics of a game (whether simulated or not), they can intuitively predict movements and results, but with less certainty. Skill and dexterity become a more important aspect of the interaction. This difference is crucial when you are using a game to educate players. ANGRY BIRDS (see figure 1.5) won a serious game award for teaching players a thing or two about physics in a fun way. While there is no doubt that ANGRY BIRDS is fun and involves physics, I doubt that players really learn about the application of forces, gravity or momentum in any conscience way that is applicable to science education. Players of ANGRY BIRDS are involved with those aspects mostly on the level of skill, rather than strategy; they might develop an intuitive feel for the effects of forces, gravity and momentum, but that is not quite the same thing as truly understanding them. Strategy in ANGRY BIRDS involves those aspects of the game that are governed by discrete rules.



Figure 1.5: In ANGRY BIRDS players catapult birds to destroy pigs protected by stone, wood and glass structures.

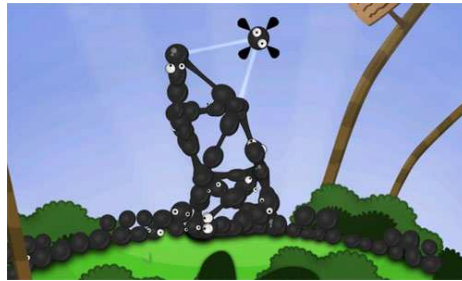


Figure 1.6: In WORLD OF GOO players construct towers, bridges and other structures from a limited supply of ‘goo balls’.

Players will have to plan how to use number and types of birds available to attack the pigs’ constructions most effectively. This requires identifying weak spots and to formulate a plan of attack, but the execution itself is based on skill and the effects can never be foreseen in great detail. Compare that to WORLD OF GOO (see figure 1.6) where players need to build constructions from a limited supply of goo balls. Physical notions such as gravity, momentum and center of mass play an important role in the mechanics of this game. Indeed, players might form an intuitive understanding of these notions from playing WORLD OF GOO. But more importantly, players learn how to manage their most important (and discrete) resource: goo balls, and use them to build effective constructions. The difference between ANGRY BIRDS and WORLD OF GOO becomes very clear when one considers the effects of continuous physics. Where in ANGRY BIRDS the difference of a single pixel can translate into a critical hit or complete miss, the effects are less felt in WORLD OF GOO. In the latter game, placement is not pixel precise: releasing a goo ball a little more to the left or right usually does not matter as the resulting construction is the same, and the spring forces push the ball into the same place. The game even visualizes what connections are going to be made before the player releases a ball (as can be seen in figure 1.6). Without trying to argue which game is more fun, I would say that players learn much more about construction in the WORLD OF GOO than they learn about physics in ANGRY BIRDS.

Physics and economy in games also affect design and innovation differently. One might say, that as games and genres evolve, the physical mechanics are all evolving into a handful of directions that correspond closely with game genres: most of the time there is little point in completely changing the physics of a first-person shooter.³ In fact, as games increasingly use physics engine middleware to handle these mechanics, there is less room to innovate in that department. On the other hand, every game is trying to create unique content, and many first person shooters do create an unique system of power-ups or economy of items to collect and consume to make their gameplay different from their competitors. If

³Although certain games, like PORTAL, have successfully introduced innovative physics systems to established genres.

there is room for creativity and innovation it is with the mechanics that govern these economies, and not with the physics of the game.

Still, looking back at four decades of computer game history, one must observe that physics has evolved much faster than any other type of mechanics in games. Physics is relatively easy to evolve because we have access to Newtonian mechanics and increasingly more computing power. The same solution does not apply to other types of mechanics. Calling all game rules ‘mechanics’ might distract developers from the fact that not all types of rules can be understood in the same way. Worse, developers might falsely assume that those other types of mechanics will turn out right, as long as we keep throwing more detailed rules and more processing power at it. The term mechanics is an unfortunate misnomer exactly because it might be holding back the development of proper understanding of different types of game rules. It might cause us to turn a blind eye to the artificial, discrete nature of those rules that are not part of the physical mechanics, but which are an equally important aspect of what makes games truly clever and unique. Without a solid theoretical framework for non-physical, discrete mechanics it is hard to evolve mechanics of that type beyond a certain point.

I will still use the term mechanics throughout this dissertation, as is customary within the game industry. However, in using the term, I will refer to the discrete mechanisms that generate a game economy more often than I will refer to continuous physical mechanics of motion. When appropriate I will differentiate between these and other types of rules, as mechanics do not impact all types of games equally.

1.3 Game Classification

What type of rules drives the gameplay of a particular game varies a lot between games and genres. Some games derive their gameplay mostly from their economy, others from physics, level progression, tactical maneuvering or social dynamics. Categorizations of games in different genres by the game industry and game journalists is usually based on the type of gameplay (Veugen, 2011, 42), and thus by extension on the different types of rules that feature more or less prominently in these genres. Figure 1.7 provides an overview of a typical game classification scheme and how these genres and their associated gameplay relate to different types of rule systems. Note, however, that this classification is one of many. There is a serious lack of consensus among the several classification schemes in use. The point here is not to present a definitive genre classification. Rather, it is to indicate how different types of rules correlate to different types of gameplay. There are many more genres and sub-genres that can be derived from this basic classification. For example, first-person shooters are a particular sub-genre of action games, whereas action-adventure games are common hybrids of the action and adventure game genres.⁴

In figure 1.7 I distinguish between five different types of mechanics. The

⁴In fact, action-adventures are so common that they constitute a separate genre in most other genre classifications.

	Physics	Economy	Progression	Tactical Maneuvering	Social Interaction
Action	Detailed physics for movement, shooting, jumping, etc.	Power-ups, collectables, points and lives	Pre-designed levels with increasingly difficult task, story-line to set player goals		
Strategy	Simple physics for movement and fighting	Unit building, resource harvesting, unit upgrading, risking units in combat	Scenarios to provide new set of challenges	Positioning of units to gain offensive or defensive advantages	Coordinated actions, alliances and animosity between players
Role-Playing	Relatively simple physics to resolve movement and conflict, often turn-based	Equipment and experience to customize a character or party	Story-line and quests to give player a purpose and goal	Party tactics	Play-acting
Sports	Detailed simulation	Team management	Seasons, competitions, tournaments	Team tactics	
Vehicle Simulation	Detailed simulation	Vehicle tuning between missions	Missions, races, challenges, competitions, tournaments		
Management Simulation		Managing of resources, economy building	Scenarios to provide new set of challenges	Managing of resources, economy building	Coordinated actions, alliances and animosity between players
Adventure		Managing a player's inventory	Story to drive game, locks and key to control player progress		
Puzzle	Simple, often non-realistic and discrete, physics generates challenges		Short levels proving increasingly more difficult challenges		

Figure 1.7: Games genres taken from Adams & Rollings (2007) and correlated to five different types of game rules or structures. The thickness and darkness of the outlines indicate relative importance of those types of rules for most games in that genre.

boundaries between these types of mechanics are not very hard, and a single game can have multiple types of mechanics. The figure indicates typical configurations of these types of mechanics as they are frequently encountered across game genres, but it should be clear that each individual game can have its own, unique configuration of game mechanics. The mechanics of physics and economy were already discussed in detail in the previous section. Progression, tactical maneuvering and social interaction are new and will be discussed below.

Progression deals with those aspects of gameplay that stems from quality level design and mechanics that control player progress through these levels. In these games, designers have created levels in which players need to overcome a predefined set of challenges. Completing a particular challenge will often unlock other challenges, and this way players progress towards a particular goal. For most of these games, the goal is to reach a particular location (where usually the final challenge awaits in the form of a “boss fight”). For this type of game, careful lay-out of the levels creates a smooth experience. They tend to take longer to complete than games that do not rely on level progression, but once they are completed, they offer little replay value: many players play through this type of game only once. Because the play experience and progress through a progression-driven game can be tightly controlled by a designer, this type of game lends itself particular well to games that also deliver stories. Typical examples of level-driven games include action-adventure games such as *THE LEGEND OF ZELDA* or *ASSASSINS CREED*, first-person shooter games such as *HALF-LIFE* or *HALO*, and role-playing games such as *BALDUR’S GATE* or *THE ELDER SCROLLS IV: OBLIVION*.

Tactical maneuvering involves those mechanics that deal with the placement of game units on a map for offensive or defensive advantages. Tactical maneuvering is critical in most strategy games, but also features in certain role-playing games and simulation games. The mechanics that govern tactical maneuvering typically specify what strategic advantages units gain from being at a particular location. These mechanics might be continuous or discrete, but discrete, tile based mechanics still seem to be common. Tactical maneuvering is important in many board games such as *CHESS* and *GO* but also computer strategy games such as *STARCRAFT* or *COMMAND & CONQUER: RED ALERT*.

Much social interaction that emerges from playing a game is not captured with mechanics. As soon as a multiplayer game allows direct, in-game interaction, social interaction outside the rules emerges. Some games include mechanics that deal with that sort of interaction more explicitly. For example, role-playing games might have rules that guide the play-acting of a character, and a strategy game might include rules that govern the forming and breaking of alliances between players.

This dissertation mostly zooms in on the discrete mechanics of economy and progression. Three reasons for this are:

1. As should become clear from figure 1.7 these types of mechanics play a role in most game genres. They are more common than tactical maneuvering and social interaction.

2. As was discussed above, there is usually more freedom for design in those mechanics that are, to a certain extent, discrete. Continuous physics generally aim to accurately simulate a real or imagined setting, the required knowledge can be taken directly from real physics. For discrete mechanics and game economy, there exist far fewer off-the-shelf solutions. This dissertation aims to contribute to the development of applied theory to improve this.
3. In order to control the scope of this dissertation, not all types of mechanics can be discussed in equal detail. Tactical maneuvering and social interaction in games are both very large topics that would warrant independent, detailed study.

1.4 Emergence and Progression

Mechanics of progression correspond to what Jesper Juul calls “structures of progression” in games which he separates from “structures of emergence” (Juul, 2002). His classification is very influential within game studies and provides a relevant framework for the study of mechanics in this dissertation. Put simply, emergence indicates that relatively simple rules lead to much variation, whereas progression indicates that many predesigned challenges are ordered sequentially. According to Juul, “emergence is the primordial game structure” (Juul, 2002, 324) that is caused by the many possible combinations of rules in board games, card games, strategy games and most action games. Games of this type can be in many different configurations or states: all possible arrangements of playing pieces in a CHESS constitute different game states as the displacement of a single pawn by even one square is a critical difference. The number of possible combinations of pieces on a CHESS board is huge, yet the rules easily fit on a single page. Something similar can be said of the placements of residential zones in the simulation game SIMCITY or the placement of units in the strategy game STARCRAFT.

Progression, on the other hand, relies on a tightly controlled sequence of events. Basically, a game designer dictates what challenges a player encounters by designing levels in such a way that the player must encounter these events in a particular sequence. The use of computers to mediate games have made this form possible. Progression requires that the game is published with much content prepared in advance, for board games this is inconvenient.⁵ As such, progression is the newer structure, starting with the text-adventure games from the seventies. In its most extreme form, the player is ‘railroaded’ through a game, going from one challenge to the next or failing in the attempt. With progression the number of states is relatively small, and the designer has total

⁵Published scenario’s for for pen-and-paper role-playing games are examples of non-digital games of progression. They take the form of books specifying setting, characters and possible storylines. However, they cannot be considered to be older forms of progression in games than computer games, as pen-and-paper role-playing originate from the same period as computer games of progression.

control over what is put in the game. This makes games of progression well suited to games that tell stories.

In the original article Jesper Juul expresses a preference for games that include emergence: “On a theoretical level, emergence is the more interesting structure” (Juul, 2002, 328). He regards emergence as a structure that allows designers to create games where the freedom of the player is balanced with the control of the designer: with a game of emergence designers do not specify every event in detail before the game is published, though the rules may make certain events very likely. In fact, a game with an emergent structure often still follows fairly regular patterns. Juul discusses the gun fights that almost always erupt in a game of COUNTER-STRIKE (Juul, 2002, 327). Another example can be found in RISK where the players’ territories are initially scattered all over the map, but over the course of play their ownership changes and the players generally end up controlling one or a few areas of neighboring territories. Despite these emerging patterns Juul acknowledges that most games combine emergence and progression. The main example in Juul’s article, EVERQUEST, is “a game of emergence, with embedded progression structures” (Juul, 2002, 327).

In his book *Half-Real*, Juul is more nuanced in his discussion of emergence and progression (Juul, 2005). Most modern games fall somewhere between games of emergence and progression. GRAND THEFT AUTO: SAN ANDREAS has a vast open world, but also a mission structure that introduces new elements and unlocks this world piece by piece. In the story-driven first-person shooter game DEUS EX the storyline dictates where the player needs to go next, but players have many different strategies and tactics available to deal with the problems they encounter on the way. It is possible to write a ‘walkthrough’ for DEUS EX, defining it as a game of progression according to Juul’s classification, but there are many possible walkthroughs for DEUS EX. Just as, at least in theory, it is possible to create a walkthrough for a particular map in SIMCITY, instructing the player to build certain zones or infrastructure at a particular time in order to build an effective city. It would be hard to follow such a walkthrough, but creating one is possible. Pure games of emergence and pure games of progression represent two extremes on a bi-polar scale, but most games have elements of both. Yet at the same time, emergence and progression are presented as two alternative modes of creating challenges in games, that might co-exists in a game, but are hard to integrate. This dissertation questions this perspective and seeks strategies to merge structured level design and emergent, rule-based play more effectively.

One trajectory towards an answer is that emergent behavior thrives somewhere on the border of chaos and order (cf. Salen & Zimmerman, 2004, 155). A true chaotic system will seem random and meaningless to most observers, whereas in games it helps if the player can make sense of what is going on. Where rules push games towards chaos by introducing dynamic behavior, levels pull games back towards order by imposing structure. If games are pulled too far back, they become games of progression where the spatial structure dominates the rules and little dynamic play remains.

This dissertation acknowledges that most games display complex, emergent



Figure 1.8: CIVILIZATION III is a good example of a game with a vast open world for the player to explore, conquer and shape.

behavior. Many games structure this behavior through level design, but some games can do without. Many casual games, such as BEJEWELED, are purely games of emergence. Many other casual games, such as ANGRY BIRDS, have many pre-designed levels that confront players with new challenges, but they are more puzzles than structured, story-like play experiences. For these games, once the mechanics are in place, many new puzzles and levels can be generated endlessly, not unlike the game of TANGRAM.

On the other hand, pure games of progression are quite rare. The most typical examples of these games are text-adventures such as COLOSSAL CAVE ADVENTURE or ZORK. But that game genre became almost extinct over two decades ago. Today adventure games are almost always action-adventure games; they almost always include some form of mechanics-driven, emergent action as part of the gameplay. So even though games can have both emergence and progression, it seems that modern games cannot do without the first, but can do without the latter.

So-called ‘sandbox games’ create an open, virtual world that is not designed to guide the player towards a particular goal. Sandbox games roughly correspond with the management simulation genre in figure 1.7. In this type of game, players are free to explore as they see fit, whether this is from a first person perspective as in GRAND THEFT AUTO III or from the god-like perspective in a game like SIMCITY or CIVILIZATION (see figure 1.8). In a typical sandbox game there are few restrictions and many optional goals for the player to pursue, some of these goals are set by the player, not the game. Will Wright, the designer of SIMCITY and many other simulation games, is quoted to have stated that his games are more like toys as they do not dictate any goals (in Costikyan, 1994). These games do not define a *variable, quantifiable outcome*. Instead, players set and value their own goals.

As a medium for telling stories or delivering a concise play experience, vast open worlds are not always the best option. Worlds have gotten so large that

the player can easily lose track of the main storyline. Hunting down the story and making your way through yet another dungeon can be experienced as quite tedious. It is a flaw that large games such as *THE ELDER SCROLLS IV: OBLIVION* or *FALLOUT 3* suffer from. It is also the reason why Chris Crawford does not put too much faith in this structure for interactive storytelling (2003b, 261-262). Yet, the artificial worlds found in games seem to grow larger and more detailed with every new release, indicating that progression remains a relevant aspect of game design.

1.5 Emergence

The use of the term emergence in games, which predates Juul's categories (for example see Smith, 2001), is often in reference to the use of the term within the sciences of complexity. There it refers to behavior of a system that cannot be derived (directly) from its constituent parts. At the same time Juul cautions us not to confuse emergent behavior with games that display behavior the designer simply did not foresee (Juul, 2002). In games, as in any complex system, the whole is more than the sum of its parts. While the active agents or active elements in a complex system can be quite sophisticated in themselves, they usually can be simulated as rather simple models. Even when the study is about the flow of pedestrians in different environments, great results have been achieved by simulating them with only a few behavioral rules and goals (Ball, 2004, 131-147). Similarly, the elements that make up games can be a lot more complex than the elements of a typical system studied by the science of complexity, but at least some games (such as *GO* and *CHESS*) are famous for generating enormous depth of play with relatively simple elements and rules. The active substance of these games is not the complexity of individual parts, but the complexity that is the result of the many interactions between the parts.

The main assumption of this dissertation is that the particular configurations of elements into complex systems that contribute to emergence in other systems also cause interesting gameplay. In other words: gameplay is an emergent property of a game system defined by its rules. For game designers this means that understanding the structural characteristics of emergent systems in general, and in their games in particular, is essential knowledge.

One of the simplest systems that show emergent behavior is a particular class of cellular automata studied by Stephen Wolfram (2002). The cells of cellular automata are relatively simple machines that abide only to local rules. The algorithm that defines their behavior takes input only from its immediate surroundings. In this particular class of cellular automata, the cells are aligned on a line, the state or color of each cell is determined by the previous state of that cell and its two immediate neighbors. With only two possible colors, this creates eight possible combinations. Figure 1.9 displays one set of possible rules (on the bottom) and the resulting, surprisingly complex pattern (on top). This pattern is created because each iteration of the system is displayed on a new horizontal line below the previous iteration. Wolfram's extensive study has revealed three critical qualities of systems that exhibit dynamic behaviors: 1)

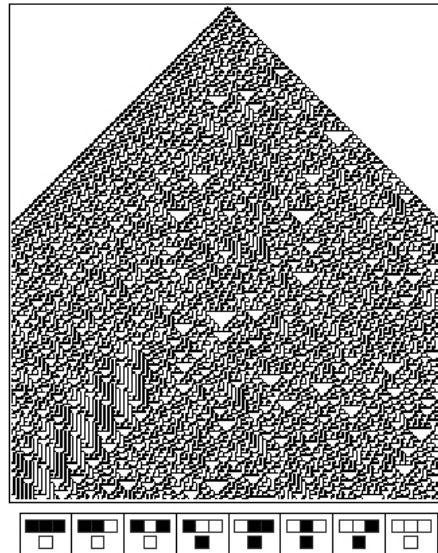


Figure 1.9: Stephen Wolfram's 'Rule 30 Automaton', taken from <http://www.stephenwolfram.com>

They must consist of simple cells whose rules are defined locally, 2) the system must allow for long-range communication, and 3) the level of activity of the cells is a good indicator for the complexity of the behavior of the system. These qualities are discussed below.

The easiest way to implement a cellular automaton on a computer is to program a simple state-machine that takes its own state and the states of its immediate neighbors as input for the function that determines its new state after each iteration. This communication of each cell's state plays an important role in the emerging behavior, without such input all cells would behave individually, and system-wide behavior would not be possible at all. In order to get more dynamic behavior communication between neighboring cells must lead to long-range communication in the system. This type of long-range communication is indirect and takes time to spread through the system. Systems that show pockets of communication with little or no communication between the pockets will show less complex behavior than systems in which such pockets do not occur or are less frequent (Wolfram, 2002, 252). Connectivity is a good indicator of long-range communication in the system. A special case of long-range communication is feedback: a cell or group of cells produce signals that ultimately feed back into its own state somewhere in the future. Long range communications travel over long distances through the system or, alternatively, through time and produce delayed effects. As we shall see throughout this dissertation, this sort of feedback is very important for games.

The number of cells that are active (cells that change their state) is important

for the behavior of the system as a whole. Complex behavior, that is behavior that is hard to predict but still seems to follow some sort of logic or hidden pattern, occurs mostly in systems with many active cells (Wolfram, 2002, 76).

Cellular automata show us that the threshold for complexity is surprisingly low. Relatively simple rules can give rise to complex behavior. Once this threshold is passed introducing extra rules does not affect the complexity of the behavior as much (Wolfram, 2002, 106).

In another study of emergence, Jochen Fromm builds a taxonomy of emergence that consists of four types of emergence (types I, II, III and IV). These types can be distinguished by the nature of communication, or feedback, within the system (Fromm, 2005). Feedback is created when a closed circuit of communication exists within a system; in effect, when a state change of a particular element directly or indirectly affects the state of the same element later on. Feedback is called positive when these effects strengthen themselves, as is the case with guitar feedback where strings are vibrated to produce sound, and amplification of the sound causes the strings to vibrate in turn. Feedback is called negative when the effect dampens itself. A thermostat is a typical example, a thermometer detects the temperature of the air, when it becomes too low it will activate a heater, the heater will then cause the temperature to rise which in turn will cause the thermostat to turn off the heater again. Negative feedback is often used in this way to maintain balance in a system.

In the simplest form of emergence, nominal or intentional emergence (type I), there is either no feedback or only feedback between agents on the same level of organization. Examples of such systems include most man-made machinery where the function of the machine is an intentional (and designed) emergent property of its components. The behavior of machines that exhibit intentional emergence is deterministic and predictable, but lacks flexibility or adaptability. Both the guitar feedback and the thermostat are examples of this type of predictable feedback.

Fromm's second type of emergence, weak emergence (type II), introduces top-down feedback between different levels within the system. Flocking is an example he uses to illustrate this type of behavior. A flock-member reacts to the vicinity of other flock-members (agent-to-agent feedback) and at the same time perceives the flock as a group (group-to-agent feedback). The entire flock constitutes a different scale than the individual flock-members. A flock-member perceives and reacts to both.

One step up the complexity ladder from weakly emergent systems we find systems that exhibit multiple emergence (type III). In these systems multiple feedback traverses the different levels of organization. Fromm illustrates this category by explaining how interesting emergence can be found in systems that have short-range positive feedback and long-range negative feedback. It propels the appearance of stripes and spots in the coat of animals and the fluctuation of the stock-market. John Conway's GAME OF LIFE is also an example of this type of emergence (Gardner, 1970).⁶ The GAME OF LIFE can easily be shown

⁶Although called the GAME OF LIFE, Conway's cellular automaton does not fall in the category of games as defined earlier in this chapter. It does not have any quantifiable goal and

to include both positive feedback (the rule that governs the birth of cells) and negative feedback (the rules that govern the death of cells). The GAME OF LIFE also shows different scales of organization: at the lowest end there is the scale of the individual cells, on a higher level of organization we can recognize persistent patterns and behaviors such as gliders and glider-guns.

Fromm's last category is strong emergence (type IV). His two main examples are life as an emergent property of the genetic system and culture as the emergent property of language and writing.⁷ Strong emergence is attributed to the large difference between the scales on which the emergence operates and the existence of intermediate scales within the system. Strong emergence is multi-level emergence in which the outcome of the emergent behavior on the highest level can be separated from the agents on the lowest level in the system. For example, it is possible to set up a grid of the cellular automata used for the GAME OF LIFE in such way that on a higher level it acts as a Turing Machine which in itself also displays emergent behavior. In this case causal dependency between the behavior displayed by the Turing Machine and the GAME OF LIFE itself is minimal.⁸

From this brief discussion a number of important observations on the nature of emergence comes forward. Within this dissertation emergent behavior is attributed to feedback loops in the system, and preferably multiple feedback loops. Only one would only lead to nominal (type I) feedback. Therefore, emergent systems must consist of multiple elements that act more or less independently. A sufficient level of activity is required; a system with only a few active elements tends to be too stable and predictable to make for interesting games. Communication (or interaction) must exist between these elements at a local scale and this local communication must indirectly enable long range communication. Feedback, a form of communication where information and actions are fed back to the source, often causes emergent behavior, especially when more than one feedback loop affects the system. Finally, emergent systems often show different scales of organization, with communication and feedback traversing these scales.

1.6 Progression

Despite the importance of emergence in games, no professional game designer can turn a blind eye towards level design and mechanics of progression. To subject yourself to game rules is to cross the boundary into the magic circle and to immerse yourself in the game's fictional space. Within that space the

does not require any effort by the player. However, as we have seen with toys, players of the GAME OF LIFE can set goals themselves, such as finding configurations that will live for a very long time, or grow into stable systems. These goals are quantifiable and do require effort to reach.

⁷One could question whether in both cases one follows from the other, or whether they have evolved in unison. Emergence might not be the best way to describe their mysteries. In fact, some researchers express serious doubts whether or not strong emergence can exist at all (Chalmers, 2006).

⁸Although this is not the same as claiming that a Turing Machine could emerge from a particular, seemingly random, starting condition for the GAME OF LIFE.



Figure 1.10: In HALF-LIFE 2 the player arrives in the game by train, but never leaves the rails.

player starts to explore the game and its possible states. The number of rules, interface element and gameplay options of a modern retail video game is usually larger than most players can grasp at once. Even smaller games found on the Internet frequently require the player to learn a multitude of rules, to recognize many different objects and to try out different strategies. Exposing a player to all these at the same time can result in an overwhelming experience, and players will quickly leave the game in favor of others. The best way to deal with these problems is to structure the game experience with clever level design that teaches the player the rules in easy-to-handle chunks. In many cases games include special tutorial levels to introduce a player to the core concepts, and even then they will introduce new concepts with extreme care.

The use of tutorials and level design to train the player is an illustration of one of the strengths of the medium of games: the use of game space to structure player experience. Unlike literature or cinema, which are well suited to depict events in time (histories), games are well suited to depict space. Henry Jenkins places games in the tradition of spatial stories, an honor they share with traditional myths and hero's quests as well as modern works by J.R.R. Tolkien (Jenkins, 2004). Simply by traveling through the game space, a story is told. A similar sentiment is found in Ted Friedman's essay on *CIVILIZATION* (1999) where the drama of that game directly stems from the player's journey through and conquest of a virtual world.

Many games have utilized this capacity to great effect. The HALF-LIFE series stands out as a particular good example. The games from this series are first-person shooter action games in which the player traverses a virtual world that seems to be vast but which in reality is confined to a quite narrow path. The

whole story of HALF-LIFE is told within the game, there are no cut-scenes that take the player out of the game, all dialog is performed by characters inside the game, and the player can choose to listen or ignore them altogether. HALF-LIFE has perfected the art of guiding the player through the game, creating a well-structured experience for the player. The practice is often referred to as 'railroading'; in this light it is probably no coincidence that in HALF-LIFE and HALF-LIFE 2 the player arrives in a train (see figure 1.10).

1.7 Approach and Dissertation Outline

Designing games for emergent gameplay and coherent progression using discrete mechanics presents designers with many problems. Perhaps the biggest handicap for the game industry is the lack of formal, theoretical tools to deal with the complexity of emergent gameplay and to assist the design of games. Several prominent designers and academics have answered Doug Church's call to develop "formal abstract design tools" (Church, 1999). This dissertation also answers that call. By developing applicable design theory for discrete game mechanics and level progression I hope to help designers understand the complex nature of game systems and to get a better grip on the elusive notion of gameplay and to create progression efficiently. In this respect this dissertation does not discriminate between board games and computer games, both are essentially rule-based artifacts and that can have emergent gameplay. The games discussed in this dissertation come from both categories in more or less equal measure.

The development of prototype software tools to assist or automate the design process is an important aspect of this dissertation. In many ways, the development of these prototypes was an important step in validation of the theory's applicability. At the same time, implementation invariably led to further improvements of the theory in what turned out to be a highly iterative process.

In the next chapter I will explore games as rule based systems in more depth. Games share some relations with simulations, which are also rule based systems. However, where simulations aim to model a source system accurately, games have a different goal: they aim to create an interesting experience. This means that games can deal with rules differently.

Chapter 3 discusses game design theories from the game industry and from academia. Although all these theories have their own merits, no theory has emerged as an industry or academic standard. In fact, some people doubt that any theoretical or methodological approach to the design of games can work, as none of these can do justice to the creativity involved in designing games. Chapter 3 will also address the arguments put forward by the people who subscribe to this opinion.

In Chapter 4 I will present my Machinations framework as an alternative theory of game design focusing on internal economy and emergent gameplay. The design of this framework takes into account the concerns that have been discussed in Chapter 3. It utilizes an abstract, visual notation to represent discrete game mechanics. The digital version of these Machinations diagrams can be run in order to simulate a game. The Machinations framework aims

to foreground those structural qualities of game mechanics that contribute to (good) gameplay.

The Machinations framework focuses on game economy and neglects level design. In Chapter 5, game levels and mechanics of progression take center stage. In this chapter I will develop the Mission/Space framework, which offers a structural perspective on these elements of game design. As an illustration of how the Machinations framework and the Mission/Space framework can be used to inform game design on a theoretical level, I will leverage both frameworks in Chapter 6 to explore how emergence and progression might be, and occasionally have been, integrated in games.

In Chapter 7 I will unite the formal perspectives on games presented in Chapters 4 and 5 under the notion of game design as a series of model transformations. Model transformation, a concept taken from software engineering, describes how models, each representing a different perspective on the subject matter, can be transformed into other models through the use of formal grammars and rewrite systems. Model transformations for game design provide theoretical leverage to automate certain aspects of the design process, as will be illustrated with the discussion of some experimental software prototypes that generate game content.

In Chapter 8 I will evaluate the applied theory and the tools developed during this research and answer the main research question: what structural qualities of game rules and game levels can be used in the creation of applied theory and game design tools to assist the design of emergent gameplay? I will discuss the result of integrating them into the game development courses taught at the Hogeschool van Amsterdam (Amsterdam University of Applied Sciences) and the workshops I have hosted at several industrial and academic conferences. The reception of the tools and theory presented in this dissertation by students, industry veterans, and academic peers is an important aspect of its validation.

1.8 Terminology

The following terminology is used throughout this dissertation:

- A *game* is a system in which players engage in artificial conflict, defined by rules, that results in a variable, quantifiable outcome affected by player effort and ability.
- *Gameplay* is an emergent property of the game as defined by its rules. Gameplay is a qualitative measure of player actions and experience; good games are said to have gameplay.
- The *mechanics* of a game is a set of rules governing the behavior of a single game element. These rules are specific: for example, a mechanism specifies exactly how fast a character moves, how high a character jumps and how much energy this costs. In this dissertation I will prefer using “game mechanism” over the slightly awkward, but commonly used, singular form of “game mechanic”. This dissertation focuses on discrete game mechanics, not on continuous physics, as these mechanics generally offer

more design-freedom and are less well understood because the reference systems available to model discrete mechanics on are less established.

- The *core mechanics* of a game are those mechanics that players interact with most frequently and which affect the gameplay the strongest. The boundaries of this set are fuzzy.
- The *internal economy* of a game is constituted by the production, flow and consumption of game resources. These resources include, but are not restricted to, power-ups, collectibles, points, items, ammunition, currency, health and player lives. These resources can be tangible or abstract. The structure of the mechanics that determine the production, flow, and consumption play an important role in the emergent gameplay of the game.
- A *level* is a particular spatial and/or logistical structure in a game that dictates what challenges players encounter. Typically, a level contains a set of positioned game elements and/or scripts to control special events and players' progress through the game.
- *Emergence* in games refers to the fact that the behavior of certain games is the result of a complex and dynamic system of rules. This means that for these games the number of possible states is huge: relatively few, and often discrete mechanics can create a large number, sometimes even infinite, of possible states. Emergence is an important source of gameplay and replay value, but it is also very hard to predict, design and control.
- *Progression* in games refers to the structures in games where a designer outlined the possible game states beforehand, usually through level design or through some form of game scripting. Progression offers much control of the play experience but has the disadvantage that it generates relatively low replay value.
- *Feedback* occurs when a change in the current state of a particular element in a system affects the state of the same element at a later time. Feedback requires that a closed circuit of causality causes the effects of the state change to 'feed back' into the new state. Feedback plays an important role in those structural qualities of game mechanics that contribute to emergence and emergent gameplay.

It seems that perfection is reached not when there is nothing left to add, but when there is nothing left to take away.

Antoine de Saint-Exupéry (1939)

2

Rules, Representation and Realism¹

Within the entertainment game industry, much effort is spent on making games more realistic. Game productions get bigger every year as graphics, physics modeling and artificial intelligence take huge strides to ever more realistic simulations. Although these developments advance our understanding of the medium of games, few developers show an active interest in alternative approaches to games. Certain type of games, such as serious games, suffer from this trend. Compared to other educational media, serious games are already quite expensive to produce. If the players and producers of these games expect a level of realistic sophistication that approaches the level found in triple-A titles these games will fail to keep up.

This chapter presents an alternative perspective on games that breaks away from realism and investigates games as a form of abstract, non-realistic, and rule-based representation. From this perspective, the strength of games does not lie in the accurate modeling of fantasy worlds but in capturing complex systems with relative simple rules, while still retaining the overall dynamic behavior of the original system a game is trying to model. In this chapter I argue that games, as rule-based systems, are excellent vehicles for knowledge, learning and entertainment, irrespective to whether these games have been created for fun or education. Even if they are not created with photo-realistic assets, detailed physics simulations and multi-million dollar budgets.

Games constitute a new form of rule-based representation that is fundamentally different from static representation through non-interactive text, images and sounds. According to Rune Klevjer, simulation is a form of procedural representation; simulation represents rules instead of events (2002). Gonzalo Frasca classifies simulation as an alternative to narrative or representation (2003, 223). Ian Bogost picks up on Frasca's work when he defines simulation as follows: "A simulation is a representation of a source system via a less complex system that informs the user's understanding of the source system" (2006, 98).

This link between games, rules and simulation is especially clear in the con-

¹This chapter also appeared in a slightly altered form as a journal article for *Simulation & Gaming* (Dormans, 2011a).

temporary focus on realism found in many modern games. Over the years games have grown increasingly more realistic. The power of modern computers allows us to render nearly photo-realistic images in real time; the visual and auditive qualities of games quickly approach the quality found in cinema. Games often refer to elements recognizable from real life: real cars, real environments, real weapons. Games that look and feel realistic sell well. In some cases the reality a game refers to is purely fictional. Games set in the Star Wars universe depict many things that are not real, but still the players have a clear idea what a Star Wars game should look, sound and feel like. Much industry research is aimed at making games more realistic. Realism features prominently in the “top ten hurdles facing game designers today” published on the website of the magazine Popular Science. All are concerned with the accurate and realistic simulation of real-life phenomena. Getting water and fire effects right made that list, as did realistic movement, rendering human faces and artificial intelligence designed to capture realistic behavior (Ward et al., 2007).

On the other hand, in certain circles of game critics and scholars, it is in vogue to point out that realism is not what games are about. Steven Poole’s *Trigger Happy* deconstructs the supposed realism of games. He argues that most players play games because they allow them to do things that cannot be done in reality. A thoroughly realistic race game, for example, would require a player to undergo thorough training before he can even try to complete a single round on a racing circuit. A game that is totally realistic ceases to be a game (Poole, 2000, 77). He concludes: “videogames will become more interesting artistically if they abandon thoughts of recreating something that looks like the ‘real’ world and try instead to invent utterly novel ones that work in amazing but consistent ways” (Poole, 2000, 240).

The sentiment that games are different from realistic and accurate simulations can already be found in the early work of Chris Crawford who states:

“accuracy is the sine qua non of simulations; clarity the sine qua non of games. A simulation bears the same relationship to a game that a technical drawing bears to a painting. A game is not merely a small simulation lacking the degree of detail that a simulation possesses; a game deliberately suppresses detail to accentuate the broader message that the designer wishes to present. Where a simulation is detailed a game is stylised” (1984, 9).²

Jesper Juul also points out that: “games are often stylized simulations; developed not just for fidelity to their source domain, but for aesthetic purposes. These are adaptations of elements of the real world. The simulation is oriented toward the perceived interesting aspects of soccer, tennis or being a criminal in a contemporary city” (2005, 172). Games allow us to do things not available to us in real life, and it is rules that grant us this power, as long as the player follows them. However, rules create both limitations and affordances. Without

²The paintings that are most stylized are modern paintings that strive to capture the essence of that which they depict through non-realistic means. I assume that Crawford is referring to this type of paintings.

rules, games would have little structure and actions would have little meaning (Juul, 2005, 58). It is for similar reasons that I linked rules to agency in a study on pen-and-paper role-playing games, even though many avid role-players tend to downplay the importance of rules in favor for interactive play-acting. In a pen-and-paper role-playing game the rules form an interface with the fictional world, and it is through rules that players can affect that world; game rules create agency (Dormans, 2006b). When looking at a game it is more important to look at what rules allow, instead of how they limit the player (cf. Wardrip-Fruin et al., 2009). Pressing the jump button in a SUPER MARIO BROS. game has the satisfactory effect of making the on-screen avatar jump way beyond the capabilities of any human being. Game rules amplify our own abilities and allow us to explore strategies or tactics in artificial conflict that would be dangerous, destructive, impractical or impossible in real life.

2.1 The Iconic Fallacy

Ian Bogost’s definition of a simulation as: “a representation of a source system via a less complex system that informs the user’s understanding of the source system” (2006, 98) closely resembles the semiotic tripartite model of the sign drafted by Charles S. Peirce. This resemblance provides an opportunity to investigate realism in games from a different, semiotic perspective. This perspective reveals that the current trend towards realism games is preoccupied with only a small subset of possible forms: iconic forms. At the same time, Peirce’s semiotic theory also suggest where to look in order to explore games and rules beyond realism.

In Peirce’s tripartite model a sign is connected to an object: that what the sign represents, and an interpretant: the mental concept the sign invokes; which “is neither an interpreter nor a user of signs” (Kim, 1996, 12). This model of the sign is best known for the classification of signs into icons, indexes, and symbols. This classification is based on the nature of the relation between the sign and its object: when a sign resembles its object it is an icon, when the sign has an existential connection to its object it is an index, and when the connection is arbitrary it is a symbol (Kim, 1996, 19-21). Figure 2.1 combines Bogost’s definition of simulation with Peirce’s model of the sign.

If games and simulation are forms of representation, then the same categories of relations between their form (simulation) and that what they represent (source system) apply to games. This perspective dictates that games, like any form of representation, always signify something outside the game. This is true even for games that are created for the purpose of entertainment only. No matter



Figure 2.1: Tripartite model of signs and simulation.

how much the “poetic function” (Jakobson, 1960) of an entertainment game calls attention to its representational form, it still is a form of communication that does refer to many meaningful and recognizable elements outside the game. Cultural interpretations of relative simple and abstract entertainment games like PAC-MAN (Poole, 2000, 178-183) or TETRIS (Murray, 1997, 143-144) have been made, and even though these interpretations are sometimes quite far fetched, the point is that, like any form of art, no game exists within a social and cultural vacuum. As David Myers points out: “human play as a cognitive and symbolic act that is fundamental to the human representational process” (1999a, 486).

In this semiotic model of games and simulation realism and iconicity are linked. We call a simulation realistic when the simulation (as a system) closely resembles the source system; we call a simulation realistic when it is iconic. From this analogy two other forms of simulation suggest themselves: indexical and symbolic simulation. If games are ultimately not realistic, then indexical and symbolic simulation might be interesting notions to help us understand games better. As we will see in the next two sections, constructions that we could call indexical or symbolic have been used in games to great effect.

Before exploring indexical and symbolic simulations I would like to push the analogy between linguistic signs and simulation one step further to make apparent an interesting discrepancy between the current focus on iconic games and the highly symbolic nature of language. Natural language is by its nature very abstract, not realistic; most words do not resemble what they stand for. And it is the abstract nature of language that contributes to language’s great expressiveness. This notion can be traced back a long time. It was already apparent in the works of seventeenth century philosopher John Locke who observed:

“Men making abstract Ideas, and settling them in their Minds with names annexed to them, do thereby enable themselves to consider Things, and discourse them, as it were in bundles, for the easier and readier improvement, and communication of their Knowledge, which would advance but slowly were their words and thoughts confined only to Particulars” (Locke, 1975, 420).

It is on similar grounds that, roughly a century later, the philosopher Edmund Burke attaches greater aesthetic power to poetry than to the realistic paintings of his age. Poets use words to “obscure” the image they try to get across. Paradoxically, this leads to a mental image that is more vivid and evocative than painting a complete and detailed picture of the same thing (Burke, 1990, 55). These days the development of abstract art has changed all this and has increased the expressive power of the image dramatically, as is exemplified by the names used by art history to identify particular genres: Impressionism, Expressionism, Abstract Expressionism, etcetera.

Ferdinand de Saussure identifies the arbitrary character of the linguistic signs as their principal characteristic. Although he does not rule out the possibility of non-arbitrary signs, he argues that in human languages most signs are arbitrarily linked to their meaning. There are usually no characteristics of what we are referring to that are connected to the words we use (de Saussure, 1983, 67-69). In

other words, language consists mostly of symbols; there are only a few linguistic icons and indexes. For Saussure too, it is the human faculty to construct a “system of distinct signs corresponding to distinct ideas” that makes language possible (de Saussure, 1983, 10). Through the human capability to take abstract meanings and handle them in bundles, human expression and understanding is taken beyond the level of particular things and into the realm of general knowledge. In other words, abstract, non-iconic presentations contain more expressive and representational power than realistic or iconic representations.

Ian Bogost’s definition of simulation quoted above is not complete. Bogost emphasizes that subjectivity is inherent to simulation: “A simulation is a representation of a source system via a less complex system that informs the user’s understanding of the source system in a subjective way” (Bogost, 2006, 98). In a simulation a system is represented through another system and the choices made in the construction of the second system reflect the values of its creator: “no simulation can escape some ideological context” (Bogost, 2006, 99). As Bogost insists, this subjectivity can be partly attributed to the fact that with simulation the simulating system is by necessity less complex than its source system. A simulating system always deviates from its source system and the choices made in that deviation reflect the understanding and/or ideology of the person or group that created the simulation. What Bogost exactly means with ‘less complex’ is not made explicit. Here, I interpret ‘less complex’ as ‘consisting of fewer parts’. The number of parts in a simulation is usually lower than the number of their counterparts in the source system. This also means that in most cases those parts are abstractions of more complex subsystems in the source system. For example the parts that make up a simulated weather system bundle many actual air-molecules that make up real weather. This makes the simulation more convenient to handle, or to paraphrase Locke: it enables us to consider the multitude of parts of a simulated system in bundles for easier and readier understanding, and for easier and readier communication and improvement of that understanding.

Thus, there always exists a gap between a simulated system and its simulation, and that gap always renders the simulation subjective to a lesser or greater extent. However, this subjectivity is the price we pay for the convenience and enhanced understanding that subjective simulations allow. In most cases the gain in expressive power outweighs the loss in resemblance to particular instances.

When one considers a simulation as essentially subjective, it is worth noting that any claim to realism becomes an ideological maneuver in itself. For example, the high level of verisimilitude in *AMERICA’S ARMY* can be read as the rhetoric claim that its apparent realism and correctness in visual representation can be extended into the ideological domain: ‘the game got its physics right, so its ethical claims must be realistic, too’ (Bogost, 2007, 78). On the other hand, in commercial entertainment games realism is often rendered as a special effect. In these games realism and authenticity becomes a spectacle designed to impress and to be appreciated by the audience. Realism, with its high poly-count, plasma effects and particle engines, is foregrounded and hyperreal. Or to use the words with which Geoff King described a very similar phenomenon in blockbuster films:

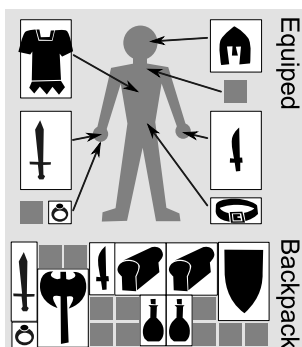


Figure 2.2: A schematic sketch of DIABLO's inventory screen.

it is “the hyperrealistic spectacle-of-authenticity rather than authenticity itself” (King, 2000, 136).

2.2 Indexical Simulation

To start looking beyond iconic simulation, the notions of indexical and symbolic simulation are obvious points of departure. In this section I will discuss the first notion, in the next section I will discuss the second. The ‘inventory system’ that first occurred in DIABLO and that has since featured in many other games can be seen as an example of the first. It inspired Warren Spector, developer of DEUS EX, into saying that: “Diablo got Inventory right. There’s no sense messing with something that works...”.³ For quite some years now, many computer games have included an ‘inventory’: the game allows the main character to pick up objects and carry them around. The player can manage these objects in the game’s inventory screen. Most games restrict the number of objects the character can carry in some way. There might be a fixed number of objects the character can pick up, or all the game objects might have a weight value attached to it and the character can only carry objects up to a particular load.

DIABLO’s inventory system takes object size as its main restricting factor (see figure 2.2). Each item takes up a number of inventory ‘slots’, the available slots are limited and organized in a grid. An item may take up 1x1, 2x2 or 1x4 slots for example. Depending on the available room in the inventory an object can be picked up or not. The upper half of the screen is dedicated to the objects the main character currently has equipped.

I argue that this is an example of indexical representation in games. The main restricting factors for somebody to carry objects in real life (shape, size and weight) are represented by easily understandable two-dimensional shapes. These

³Quoted on S. T. Lavavej’s Deus Ex webpage. URL: <http://nuwen.net/dx.html> (last visited June 23, 2011). However, it must be said that opinions on this system differ; not everyone is as enthusiastic as Warren Spector (see for example Adams & Rollings, 2007, 516-518). Whether it is a good design decision to burden players with the upkeep of their inventory depends on the type of game and intended gameplay.

shapes and their relative size can be said to be existentially connected to the size and weight of their simulated counterparts. Therefore the simulation qualifies as an indexical construction as it is parallel to indexical signs in which the relation between the sign and its object is also based on an existential connection (rather than resemblance or arbitrary convention).

The number of games that have copied this system in one form or another is a testimony to the quality of this construction. The internal rules and constraints are immediately apparent (not in the least because they are tailored towards visual representation on a screen). The management problems the system gives rise to are very much like those problems in real life. The system even allows players to make an inefficient mess of their inventory, teaching them something about the need to organize themselves.

What the DIABLO inventory system does very effectively is to take many related and similar functioning game rules and replace them all by a single mechanism that is well suited to the medium of the video game. Obviously some accuracy of simulation is lost (an item cannot be large and light at the same time), but the overall behavior is retained (the players are limited in what they can carry). The cleverness of the DIABLO inventory is that it collapses all the nuances of managing an inventory into a problem of size, which is easily represented by a computer screen, instead of weight which was the more common choice before, but which translate to the visual medium of the computer less well.

Another example of indexical simulation is the way most games handle 'health'. Health of characters and units is often represented by a simple metric, be it a percentage or a number of 'hit points'. Obviously in real life the physical health of a person or the structural condition of a vehicle is a complex matter to which many different aspects contribute. By using a generic health for a single character games bundle all these aspects into one convenient mechanism. Both players and computers can easily work and understand the numerical metric to represent the bundle.

2.3 Symbolic Simulation

Symbolic simulation goes one step further in breaking away from modeling a system with rules that closely resemble the mechanisms of the source system. The use of dice in many board games tends to be symbolic. For example, the roll of a few dice can stand for a complete battle in a game of RISK. In this case, the relation between rolling dice and fighting is arbitrary, and one simple action well-known from other games is used to simulate a multitude of actions for which most players would lack expertise. Dice can replace these battles because, for the purpose of the game, as the player should have little influence over the outcome of these battles. RISK is about global strategy, not about tactical maneuvers on the field of battle. A player cannot control the result of dice (not without cheating anyway) just as a supreme army commander cannot win every battle personally. Yet, the player needs some sort of influence and the rules tie in with the dice rolling: committing more armies to a battle allows the player to roll more dice and improve chances for success. Something similar



Figure 2.3: KRIEGSSPIEL is played by maneuvering units around a real map, combat is resolved using dice.



Figure 2.4: Jumping to avoid or defeat enemies in SUPER MARIO BROS.

goes for KRIEGSSPIEL (figure 2.3) and many successive war games. In contrast to RISK, these games *are* all about tactical maneuvering on a battlefield. So the rules for these maneuvers are quite elaborate. But the rules covering actual fighting are left to dice and attrition tables. Again, these games are to train tactical skills, not how to use a gun.

Dice are wonderful devices to create a nondeterministic effect without the need of detailed rules. From a suitable high level of abstraction, a complex and nondeterministic system, such as fighting, has a similar effects as rolling a few dice. Especially when the player is not supposed to have much influence over this system, dice mechanics can be used to replace the more complex system. The characteristic randomness of different dice mechanics can be used to match many superficial, nondeterministic patterns created by more complex systems. Pen-and-paper role-playing games have come up with many clever and interesting ways of using dice, allowing more or less influence by the player. In fact, dice mechanics related to a set of characteristics representing skills and attributes forms the core of most pen-and-paper role-playing systems. Often the same mechanism is used to represent a wide variety of actions.

Other examples, such as jumping on top of enemies in order to dispose them in the classic video game SUPER MARIO BROS. fall somewhere in between symbolic and indexical forms of simulation (see figure 2.4). Although the precise implementation differs from enemy to enemy, and certainly does not work against all enemies, it is a frequent feature throughout the game and the series it belongs to. It is unlikely that I am the first to point out that this method is a little odd, to say the least. However, it has become a convention within platform games that is instantly recognizable to gamers, and ties in with that genre's defining action of jumping from platform to platform.

The connection between jumping on top of something and defeating something in real life is not completely arbitrary, but its use in platform games has become so conventional it parallels the definition of a symbolic sign in language. In the real world, there are creatures that can be squashed by jumping on top of

them. However, there is no creature that I know of that is lethal when bumped into, but not when stepped upon, which makes this exact mechanism somewhat arbitrary.⁴ What is more, this method of fighting in *SUPER MARIO BROS.* is motivated more by the use of the genre's most prominent action of jumping, than it is motivated by any claim to realism. The link between the simulation and what is simulated is both arbitrary and conventional. Especially in the multitude of platform games that followed the example set by *SUPER MARIO BROS.*

There is, however, an affinity between the skills needed to defeat enemies in *SUPER MARIO BROS.* and in real life. In the game, it requires timing and accuracy, which are among the skills involved in real fighting. The point is, the simple representation in the game allows us to do more than to hone and train those skills. The simple metaphor of jumping on top of enemies is easy to grasp by the player, but the game then goes on by inviting the player to experiment and develop strategies. In most platform games each level ends with a 'boss' enemy which is typically designed to test the effectiveness of players' strategy. It is the ultimate test for the players to demonstrate they understand and have mastered the simulation, and are able to combine different moves.⁵

What the jumping on enemies mechanism accomplishes is a very clever way of adding combat rules to a jumping game; it introduces no new actions for the player. It manages to do this by replacing actions it tries to represent by other, arbitrary rules already implemented in the game. This reduces the number of actions players need to learn, allowing players to quickly move on to a deeper, more tactical or strategic interaction with the game instead of fussing around with its interface. As is argued below, symbolic simulation effectively reduces the system to a simpler construction with more or less equivalent dynamic behavior.

2.4 Less Is More

Indexical and symbolic simulation tend to create simpler game systems than iconic simulation. The reduction in rules these forms of simulation allow is in general benevolent. Simpler games are easier to learn, yet they still can be quite difficult to master. Games are not the only medium for which the expression 'less is more' rings true. In almost any form of representational art saying more with less means is appreciated, especially by critics and connoisseurs. Christopher Alexander, drawing inspiration from poetry for his pattern language for architecture and design puts it like this:

“This language, like English, can be a medium for prose, or a medium for poetry. The difference between prose and poetry is not that different languages are used, but that the same language is used differently.

⁴Even if such creatures do exist, they are certainly are not tortoises.

⁵These lessons carry over to situations beyond the game. The mentality of the players that have learned these lessons is excellently described by John Beck and Mitchell Wade: they know that solutions will eventually present themselves, and they have mastered a trial and error approach to many problems in life (2004, 11-14).

In an ordinary English sentence, each word has one meaning, and the sentence too, has one simple meaning. In a poem, the meaning is far more dense. Each word carries several meanings; and the sentence as a whole carries an enormous density of interlocking meanings, which together illuminate the whole.” (Alexander et al., 1977, xli)

And:

“It is essential then, once you have learned to use the language, that you pay attention to the possibility of compressing the many patterns which you put together, in the smallest possible space. You may think of this process of compressing patterns, as a way to make the cheapest possible building which has the necessary patterns in it. It is, also, the only way of using a pattern language to make buildings which are poems.” (Alexander et al., 1977, xliv)

For poetic language, or rather for any form of representation art, this quality is very important and does not stem from the use of abstract signs only. The combination and structure of these signs, or to use the linguistic term, syntactical relations between these signs also play an important role. In this light, Noam Chomsky observed that language allows speakers to make infinite use of finite means: the number of words we have may be limited (and is vastly outnumbered by particular things in reality), the number of combinations we can make with them is infinite (Chomsky, 1972, 17). This characteristic of language is often called discrete infinity.

It is impossible to exactly quantify how many rules a game should have; it is impossible to quantify how much less is how much more. Each individual design has its own balance. A particular number of rules could be too few for one game and too many for another. The balance a game should seek to strike is between the number of gameplay options the rules create on the one hand and the cognitive burden it requires to understand or operate those rules on the other. Antoine de Saint-Exupéry’s famous quote “it seems that perfection is reached not when there is nothing left to add, but when there is nothing left to take away” (1939) applies extremely very well to games.

In general, games are very good at creating endless possibilities with only a few rules. It is estimated that there are more possible game states in games like CHESS and GO than there are atoms on earth (see Shannon, 1950). It is the rules of the game that determine the number possible states, but it is not necessarily true that more rules will lead to more possible states. In addition, when a game can create a large number of possible states without using many rules, the game will be more accessible.

Possible game states and trajectories through a games state space are emergent properties of the game rule system. The elusive notion of gameplay is related to these properties. Games that allow many interesting trajectories arguably have more gameplay than games that generate fewer trajectories or less interesting ones. However, determining the type and quality of the gameplay is hard, if not impossible, by simply looking at the rules. Comparing the rules



Figure 2.5: In CONNECT FOUR gravity makes sure players can only occupy the bottom most, unoccupied square in each column.

of TIC-TAC-TOE and CONNECT FOUR serves as a good illustration of these difficulties. The rules for TIC-TAC-TOE are:

1. The game is played on a three by three grid.
2. The players take turns to occupy a square.
3. A square can only be occupied once.
4. The first player to occupy three squares in a row (orthogonally or diagonally) wins.

The rules for CONNECT FOUR are (with the differences emphasized):

1. The game is played on a *seven by six* grid.
2. The players take turns to occupy a square.
3. A square can only be occupied once.
4. *Only the bottom most unoccupied square in a given column can be occupied.*
5. The first player to occupy *four* squares in a row (orthogonally or diagonally) wins.

While the differences in rules for these two games are only a few, the differences in gameplay are immense. Far larger than the difference in cognitive effort needed to understand the rules. In the commercially available version of CONNECT FOUR, the most complicated rule (number 4) is enforced by gravity: a player's token will automatically fall to the lowest available space in the upright playing area (see figure 2.5). This relieves players from manually enforcing this rule and allows them to focus on the rules effects instead. Despite the small difference in the complexity of the rules, TIC-TAC-TOE is suited only for small children, whereas CONNECT FOUR can also be enjoyed by adults. The latter

game allows many different strategies and it takes a considerable longer time to master the game. When two experienced players play the game, it will be an exciting match, instead of a certain draw as is the case with TIC-TAC-TOE. It is hard to explain these differences just by looking at the differences in the rules.

These days, emergence of complex behavior from relatively simple elements is an important aspect of many fields of research in the domains of mathematics, physics and social sciences. In the research of games, too, emergence is becoming an increasingly important notion. From the computational side, emergence is an important technique used in anything from development of artificial intelligence to the realistic rendering of water and fire. For Penelope Sweetser the disadvantageous loss of creative control in a system that is set up for emergence is outweighed by the more consistent and intuitive player interactions such systems allow (Sweetser, 2006, 14). Likewise, game designer Harvey Smith argues that attempting to design a totally controlled game environment that allows rich interaction is no longer economically viable, as the sheer amount of detail cannot be efficiently produced manually (Smith, 2001).

One major advantage of games that feature emergent gameplay is that a rule-system allows, and often even invites, players to experiment with the game, instead of merely repeating the moves a game designer intended. Ultimately emergent games allow the transformation of the game rules itself (Myers, 1999b). This has severe consequences when building an educational game, but also when the game designer has a particular story or message in mind. For Jan Klabbers it is the responsibility of the game designer to shape the whole of the game system in such a way that behavior that conforms the design specifications emerges from its components. At the same time, the system should leave enough freedom for players to act according to their own strategies, goals and incentives, in order to elevate the position of the player into that of a reflexive actor. This is “one of the major bottle necks in the design” (Klabbers, 2006, 102).

However, in some ways, computer games seem to be moving against the trend of emergence. Jesper Juul differentiates “games of progression” from “games of emergence” as a historical newer category associated with computer games. The rise of computer games, and adventure games in particular, has made games of progression possible, as without a computer the amount of data and the number of special case rules facilitating the progression through a multitude of game spaces would have become unwieldy (Juul, 2005, 5).

Chris Crawford’s notions of data intensity and process intensity (Crawford, 2003a, 89-92) can be pitted against Juul’s observation that games of progression are a younger form and the implication that progression is the result of a natural evolution of the medium. Crawford argues that computers are both suited to handle large amounts of data and crunch vast quantities of numbers, but it is the latter ability that sets computers apart from most other media. Handling data is something that all media are good at. The computer often allows faster access to remote locations within the data, an ability put to good use within hypertext (Lister et al., 2003, 23-30). However, it is the ability to create new content on the fly where the computer really shines. Like no other medium before, the computer has the capacity to surprise players and designers alike (see also Smith, 2001).

For Chris Crawford, games should capitalize on this ability of the computer, games should be process-intensive, rather than data-intensive. In other words, games should be games of emergence rather than games of progression.

2.5 Designing Emergence

Designing emergence is a notoriously hard, somewhat paradoxical, task. Emergent properties of a system only surface when a system is put into motion. Even when a system behaves in a certain way during all test, there is no guarantee it will do so all the time. In this light the realistic fallacy seems to be a fairly conservative strategy to avoid the difficulties of designing truly emergent games: that have relative simple systems and display interesting, complex behavior. Simply adding more, and more detailed rules is only a poor substitute for creating complex gameplay through a lean and elegant rule system.

Emergence can be the result of relatively simple rules, therefore games do not need to rely on complex rule systems in order to create interesting gameplay. On the contrary, using simple means to generate complex gameplay has many advantages. The design becomes easier to manage for the designer, and the game becomes easier to learn for the player. In the examples of non-iconic simulation above (DIABLO's inventory, the use of hit points, dice in KRIEGSSPIEL and jumping in SUPER MARIO BROS.), the use of indexical and symbolic simulation resulted in a simpler rule system than an iconic simulation would have. This is not a characteristic of the examples discussed above, rather it is the advantage of using non-iconic rules in games. Compared to a completely detailed, realistic system that tries to simulate through accurate detail, indexical and symbolic simulation aims to capture the essence of the source system with fewer means. When done correctly, the result is a leaner, more elegant system that minimizes on parts and maximizes on expressiveness.

In essence, indexical simulation bundles a number of related and more or less isomorphic rules into one game mechanism. Symbolic simulation goes one step further, it connects rules in the game where they would not be connected directly in the source system. As in the use of symbols in language, there are symbols that work better than others. The symbols that work best seem to connect two unrelated rules that still have some affinity between them. In the case of SUPER MARIO BROS. there is a natural affinity between the physical skill and timing involved in both jumping and fighting (also see Lakoff, 1987, 448).

The development of the serious board game GET H2O, in which I took part, is very good example of the application of non-iconic reduction. In this game, produced as part of an educational program for adolescents in East-Africa, the players struggle to survive in the poor residential areas of an African metropolis (see figure 2.6). The vital resources are scarce, players need to balance carefully between personal gain and community efforts. The players only have indirect influence over bad events that might happen, but sometimes players can benefit from these events, sowing the seeds for conflict. The game simulates life in an African metropolis, and is designed to give the players a top-down view of their own lives. It is designed to function as a vehicle for exploration, discussion and



Figure 2.6: A prototype of GET H2O being played in Nairobi (photo by Butterfly Works).

reflection.

Instead of trying to simulate the East-African urban life in detail, the game reduces the number of resources and rules to a relative simple set. The game uses three main resources: money, houses and clean water. The latter two determine how many actions a player can take each turn while the money is used to build more houses. These resources are under constant threat. Money and water might get stolen, houses might be burned down. In reality, an African family has many more needs, but these three resources are enough to simulate an economy of scarcity that behaves not unlike the real economy in urban African areas. The indexical nature of this simulation makes it possible to create a relatively simple system that is still recognizable for people who grow up in those areas. In fact, the game became more recognizable because it lacked detail: fewer details creates more room for personal interpretation and less chance that the game does not match the individual experience. What is more, this economy creates the particular balance between short term personal gain and long term community interests that causes social instability. Creating this instability was the prime design goal for the game. The game is supposed to train people in dealing with such a situation in the first place. Simply put, more resources and a more complex economy were not needed to replicate the volatile social system of an East-African urban sprawl.

The game also uses symbolic simulation. After every player has taken a turn, all players discard one playing card without revealing it. These cards are normally used for player actions. The discarded cards are then shuffled and revealed. Every card has a symbol representing bad things that might happen, from corruption and pollution to arson and drought. If the same symbol is played

twice the effects are aggravated: one drought symbol does nothing, but two drought symbols indicate that a drought strikes, often with devastating effects. Obviously playing cards have nothing to do with the occurrence of real droughts. The cards are a way of simulating bad events that are mostly beyond the control of the people living in an African metropolis. One that also conveniently ties in with other mechanics of the game: all players will also get a secret role which allows them to benefit from bad events such as corruption, scapegoating and arson.

When used correctly, indexical and symbolic reduction reduces the number of elements in a system without affecting its structural complexity and emergent properties too much. In the GET H2O example many similar resources that are needed on a daily basis are replaced by just one: water. The feedback structure that entails having access to these resources is pretty much unchanged. The number of feedback loops, for example, is not affected. In fact, the game emphasizes these structural features, by taking away unnecessary detail. By reducing the number of elements in a game system the cognitive burden of the player in keeping track of all these elements is also reduced, allowing the player to focus more on these features and the strategic interaction that they allow, or in the case of GET H2O, on the social implications they have.

There are more advantages. A system that uses indexical and symbolic simulation can concentrate the experience, allowing a complete session of play to run much quicker than what the play represents in real time. The player is confronted with the results of his actions fast and efficiently. It allows players to 'handle the rules in bundles for the easier and readier improvement of their understanding of the system'. On the one hand this allows players to go through the process more often and on the other hand it will contribute to the pleasurable experience of agency and power that drives many commercial entertainment games. In the GET H2O game this was certainly one of the design goals. The game can be played in roughly forty-five minutes, allowing players to experiment with different strategies efficiently while reducing the costs of failure.

For the designer of games there are advantages, too. A game system that is reduced to its essence becomes better manageable and easier to balance. Without many parts, the designer can focus on those elements and structures that contribute directly to the game's emergent behavior and more easily tweak that behavior into the desired shape. Games would do well to strive for non-iconic, discrete infinity rather than detailed realism. Not only is this economically more feasible, it is also more interesting artistically and it allows for more effective communication.

THE LEGEND OF ZELDA series is a great example of gameplay design in which only a handful of game objects and associated rules are combined in many interesting challenges. The value of each of these objects and their rules does not stem from its power to represent some sort of realistic aspect of adventuring through a dungeon, but from a potential combination with other objects and rules. The exploration challenges, which the series is famous for, are almost always the result of combinations of simple, reusable gameplay mechanics that are often quite indexical or symbolic. For example, in the THE LEGEND OF

ZELDA: TWILIGHT PRINCESS the player can find the ‘gale boomerang’ in the ‘Forest Temple’ level, which creates a gust of air strong enough to activate wind-operated switches and carry small items to the player. This boomerang can be used to carry ‘bomblings’, little creatures that explode a few seconds after the player grabs him by hand or with the gale boomerang. Effective use of this combination is required to defeat the final boss in that level. The same boomerang is used in THE LEGEND OF ZELDA games for the Nintendo DS, but in this case the player can use the stylus to draw the path of the boomerang quite freely, directing it around obstacles unrealistically. Players appreciate this sort of structures as they have the advantage of being inheritably coherent. And as has been pointed out before, coherence is a strong contributing factor to gameplay (Poole, 2000, 64-66). One can even argue that the appreciation of such structures is in its essence an aesthetic appreciation (Huizinga, 1997, 25). It is the appreciation of the craftsmanship of the game designer in building systems with interesting structural qualities from which interesting behavior emerges. It forces to pay attention to the way the game was constructed and how it is structured (cf. Ryan, 2001, 176).

The meaning that emerges from these games is not necessarily less detailed or less valuable than games that aim for detailed and realistic simulation. On the contrary, as the challenges of exploration in THE LEGEND OF ZELDA are more abstract, the skills and knowledge the game addresses are more generic; the message of a game that is less iconic is much better applicable outside the particular settings of the game. This is especially useful when one wants to express something through a game that has value beyond the game and its immediate premise.

2.6 Conclusions

Games and simulations share a representational form: representation of source system through a system of rules. Even games that are played as a pure form of entertainment simulate something. Rule-based representation can take many different forms. Traditional simulation has accurate modeling as its main goal, whereas most games are designed to entertain. Games that aim to educate can be said to fall somewhere in between: they seek a certain level of accuracy, but generally enjoy more design freedom than simulations do. However, many games still conform to the norm of simulations; they aim to represent a source system by creating rules that resemble the rules of the source as closely and accurately as possible. This type of rule-based representation is called iconic simulation, in analogy to general semiotics.

Also in analogy with semiotics, the notions of indexical and symbolic simulation were explored as possible avenues to differentiate games from simulations. As pointed out above, the goal of a game is not the same as the goal of a simulation. Indexical simulation, where the rules of the game have some sort of causal relation with the rules of the source, and symbolic simulation, where the rules of the game are linked to the source by convention, allow for much simpler game systems. Although these systems consist of fewer rules and parts, their behavior

or meaning need not be less complex. The power of non-iconic simulation, such as games, lies not in its power to accurately model a source system or in the creation of a vast, realistic game world, rather in its efficient use of expressive game mechanics. I do not wish to claim that the potential of iconic representation has been fully explored. However, to me it is clear that much more progress can be made by developing indexical and symbolic building blocks for simulation, and, more importantly, investigate the effectiveness of particular configurations of such building blocks. Emergent behavior is more likely to originate from the interrelations of game parts than simply the parts themselves. It is the craft of the game designer to create complex systems from appropriate and simple elements. There is little art in creating complex simulation with equally complex (or worse, more complex) means. Yet this seems to be what many developers aim for.

Indexical and symbolic simulation as discussed in this chapter are suggestions to go beyond iconic simulation. They are theoretical notions that help reduce a game system to its bare minimum without affecting the structure from which the gameplay emerges too much. This allows the designer to focus on balancing the emergent behavior and provides the player with a better opportunity to explore the ludic significance, or generic knowledge, codified by the game.

Yet, to design games with emergent gameplay is by no means an easy task, even when the rules are kept simple. Emergent behavior is by definition unpredictable. A game designer's best bet is to create many prototypes and keep testing them. But even with frequent tests, game designers have to rely on their experience and intuition to create their games. This practice can be improved by creating better tools for the job, especially design tools that acknowledge the emergent aspects of games and focus on those structural qualities that drive them. In this light, Chapter 3 will discuss previous efforts and existing design tools while Chapter 4 will present the Machinations framework as a new, alternative scheme to deal with game mechanics and structures of emergence in games.

He that would perfect his work must first sharpen his tools.

Confucius

3

Game Design Theory

The previous chapters focused on the nature of games. This chapter discusses available tools and theory to assist developers designing games. A number of design guidelines, methods, theories and tools have been developed over the past years. Some of these were developed specifically to assist the design process, while others were developed as analytical tools, work methods, or documentation techniques. The main approaches and attempts to assist game designers that have been developed up until now and that are discussed in this chapter are: design documents, the MDA framework, play-centric design methodologies, game vocabularies, design patterns, finite state machine diagrams, Petri nets, and finally different types of game diagrams. Most of these methods were not developed as design tools, yet all of them can be used as such or might have an impact on the development of new design methods and tools. This chapter discusses the merits of all these methods for the purpose of developing design methods and tools.

The reader should note that the common discourse about these methods is quite diffuse. Within the game industry, and to a lesser extent within game research too, there is no fixed vocabulary. Many concepts are used quite informally, and terminology frequently overlaps or even conflicts. For example, the term ‘game design document’ captures a wide variety of different documents that are created for as many different reasons. Furthermore, there seems to be little distinction between analytical methods and design methods, and the two terms are sometimes used interchangeable. I have tried to use the original terminology as much as possible. Hopefully I have done so without creating confusion.

In addition, not everybody in the game industry sees the benefits of methodological approaches to game design, such as the ones discussed in this chapter and presented in the following chapters. The two most common arguments against design methodologies are that they have little practical value for game design and that they cannot replace the creative process of designing games. I will address these arguments in the last section before drawing a conclusion.

3.1 Design Documents

Almost every game company creates design documents. On the Internet many different templates can be found that are used by different companies, and virtually every book that discusses game design has its own template. The notion and practice of design documents is as diffuse as it is diverse. There are many different reasons to write these documents, and there are many different moments in the design process in which companies do so. Game design documents are sometimes used to record designs before they are built, and sometimes they are used to record designs after the games have been built. They typically contain descriptions of a game's core mechanics, level designs, notes on art direction, characters and their backgrounds, etcetera. Some advocate lengthy detailed descriptions covering every detail of a game, while others favor brief documents that capture design targets and design philosophy.

Over the years, writing game design documents has become a common industry practice, although no standard emerged that describes how, when or to what purpose these documents should be written. It is not uncommon to produce an entire set of documents, each focusing on a different part of the design or facilitating a different stage of the design process (see for example Adams & Rollings, 2007; Rogers, 2010). Without a widely accepted template for design documents, they do not carry over from company to company or from university to professional career. Without a widely accepted template, design documents cannot grow into a standard methodology. The fact that most design documents have their own style and use their own unique concepts to describe games does not help to create a generic body of knowledge beyond the scope of each individual project or company. With no industry wide standard in sight, it is unlikely that design documents are going to be effective in the near future.

What is more, design documents might not be the right tool to deal with the dynamic, emergent behavior of games. Game design documents that are written before any prototype is made are the equivalent of requirements documents in software engineering. A requirements document lists the requirements and functionality of a new, custom-built software application. Its creation is one of the first steps in the 'waterfall method' of developing software, in which each step is completed before proceeding to the next step. This document is typically written before the software is built and frequently is part of the agreement between contractor and client. The waterfall method assumes that all requirements are known and can be recorded before the software is built. Within software engineering creating and documenting functional designs is a time-tested practice, although, with the recent popularity of agile development methods, the practice of writing complete functional designs as a blueprint for a new software application has lost its appeal.

There are three important differences between designing games and business applications using a waterfall method that make it difficult or inefficient to transfer the practice from general, custom software development to game development:

1. Game design is a highly iterative process; no matter how experienced a designer is, chances are that the design of a game is going to change as it is being built. Due to the emergent nature of games (see chapter 1) it is often impossible to accurately predict the behavior of a game before it is implemented. In games changes to the implementation are to be expected.¹
2. Not all games are created within a contractor-client context. This is especially true for entertainment games, which development shares more with commercial, off-the-shelf software development. Without this context there is less need to document the design before the game is built. Although publishers and funders for entertainment games set goals and milestones, they do not function in the same way.
3. Games are rarely built with upkeep or future development in mind, reducing the necessity to create documentation that aids future developers. Despite the fact that many sequels are produced in the game industry, many sequels are built from scratch, surprisingly little code is being reused. From the perspective of software engineering this is a bad practice. However, as the development techniques are still evolving fast and a new generation of hardware becomes available roughly every five or six years this practice makes more sense from the perspective of the game industry.²

Many designers regard the game design document as a necessary evil, and some have dismissed the practice entirely (Kreimeier, 2003). Everyone agrees that designs need to be documented and communicated to the team, but in practice people hardly look at design documents (Keith, 2010, 85-87). Stone Librande, creative director at Electronic Arts, experimented with a technique he calls one-page designs to circumvent some of these problems (2010). His approach is to create design documents that are more like data visualizations instead of multi-page, written texts. One-page design documents are posters that capture the essence of a game visually (see figure 3.1). These documents have four advantages over the traditional design documents:

1. Most designers find them more interesting to create, making the task of creating a design document less tedious.
2. Because there is a spatial constraint (although the size of the page is left to the whims of the designer), the designer is forced to focus on the essence of the game. This makes the document a better match for the agile development process often found in games.
3. As people tend to like the way these documents look, they tend to stick them to walls, increasing their exposure and impact.

¹Although, it must be noted that this is also increasingly true for business software.

²The developers of reusable game components that are sold to game studios ('middleware') are probably the exception to this rule. They do maintain and reuse their code, but as their core business is not developing games, this practice does not carry over to game design.

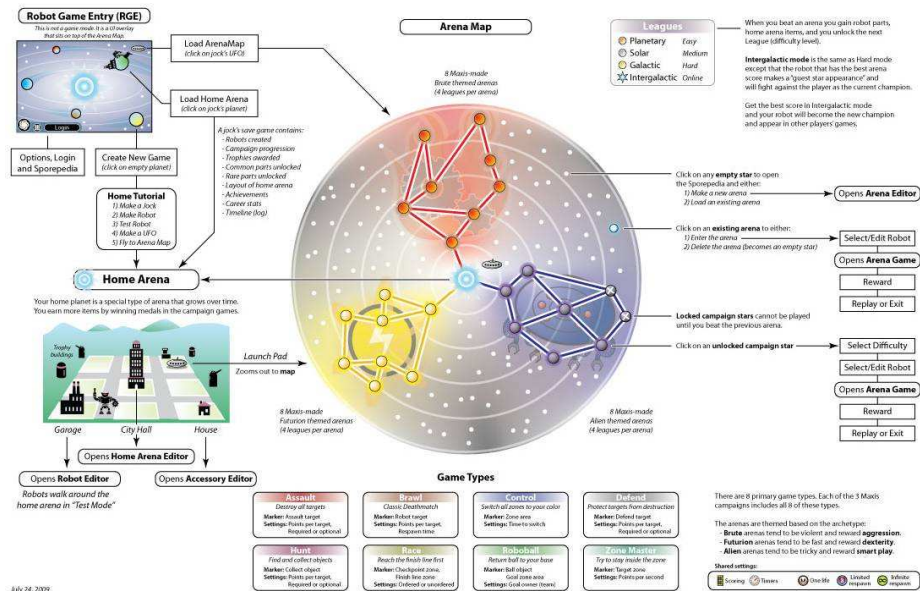


Figure 3.1: A sample one-page design document from Librande (2010).

- Stone Librande suggests leaving plenty of whitespace on the documents in order to invite team members to scribble notes on them. This keeps the documents up to date.

One-page design documents solve some of the problems associated with design documents, but not all. There is still no standard for documenting gameplay, mechanics and rules. Each one-page design document is created for a particular game, and although the product should be understandable and communicative, it cannot set a standard. In addition, the lack of detail, which makes a one-page design document more flexible and therefore is one of its strengths, makes it less suited to record designs; one-page design documents are very good at capturing the design vision, goals and direction, but they cannot function as a technical blueprint at the same time.

3.2 The MDA Framework

The MDA framework, where MDA stands for *mechanics*, *dynamics* and *aesthetics*, has been used to structure the game design workshops at the Game Developers Conference (GDC) for at least eleven years running (from 2001 to 2011).³ In contrast to the practice of game design documents, the MDA framework quite consciously tries to present a generic approach to the difficulties

³Unfortunately in software engineering the same acronym is widely used to denote Model Driven Architecture, this might lead to some confusion. In this dissertation MDA always stands for the mechanics, dynamics and aesthetics framework.

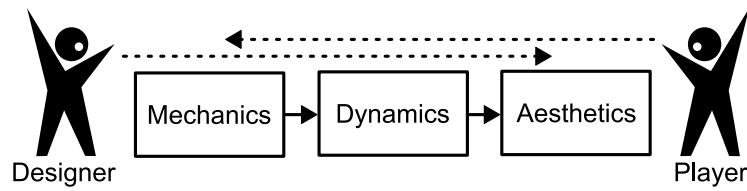


Figure 3.2: The MDA framework (after Hunicke et al., 2004).

involved in designing games. It has been quite influential and it seems to be one of the most frequently recurrent frameworks found in university game design programs all over the world. It probably is the closest thing the industry has to a standardized game design method.

The MDA framework breaks down a game into three components: *mechanics*, *dynamics* and *aesthetics*, which correspond to the game's rules, its system, and the fun it brings (Hunicke et al., 2004). The MDA framework teaches that designers and consumers of games have different perspectives on games. Where a consumer notices the *aesthetics* first and the *dynamic* and *mechanics* afterwards, a designer works the other way round. A designer creates *mechanics* first and builds *dynamics* and *aesthetics* on top them (see figure 3.2).

The MDA framework is designed to support an iterative design process and to help designers to assess how changes in each layer might affect the game as a whole. Each layer has its own design goals and effects on the game. *Mechanics* determine the actions a game allows and it affects the game's dynamic behavior. The *dynamics* layer addresses concepts such as randomness and complexity to explain a game's behavior. Finally, the *aesthetics* layer is concerned with the game's emotional target: the effect it has on the player. The MDA framework describes eight types of fun as prime aesthetic targets. The eight types of fun are: sensation, fantasy, narrative, challenge, fellowship, discovery, expression and submission (Hunicke et al., 2004; LeBlanc, 2004).

Despite the influence of the MDA framework and the long running GDC game design workshop, as a conceptual framework the MDA never seems to have outgrown its preliminary phase. The distinction between the *mechanics* and *dynamics* layers is not always clear, even to the original authors (see LeBlanc, 2004). The *mechanics* are clearly game rules. But the *dynamics* emerge from the same rules. Yet, the original MDA paper places game devices such as dice or other random number generators in the layer of the *dynamics*. To me, those devices would seem more at home in the layer of the *mechanics*. Likewise, the *aesthetics* layer seems to contain only the player's emotional responses. The visuals and story that cue these responses, which would commonly be understood as being part of an aesthetics, seem absent from the framework. The eight kinds of fun comprise a rather arbitrary list of emotional targets, which is hardly explored with any depth. Apart from short one-sentence descriptions, Hunicke et al. do not provide exact descriptions of what the types of fun entail. They do state their list is not complete, but they do not justify why they describe these eight, or even hint at how many more types of fun they expect to find. What

is more, the whole concept of ‘fun’ as the main target emotion of games has been criticized by Ernest Adams and Andrew Rollings (2007, 119), and Steven Johnson (2005, 25-26), among others. Games can aspire to target a much wider variety of emotional responses. Some additional MDA articles (such as LeBlanc, 2006) have appeared over the years but they have not taken away these concerns.

3.3 Play-Centric Design

A more thorough method of iterative game design is described by Tracy Fullerton et al. (2006). Coining the term ‘play-centric design’ to describe their method, Fullerton et al. advocate putting the players at the heart of a short design cycle. They advise that game prototypes are built quickly and tested often. Because of the short design cycle more innovative options can be explored with a reduced risk measured in effort and time.

Play-centric design distinguishes between two levels in a game: the formal core and a dramatic shell surrounding it. A game’s formal core consists of rules, objectives and procedures whereas the dramatic shell consists of premise, character and story. Combined, these two layers contribute to the dynamic, emergent behavior that supports play. It is the objective of play-centric design to tune this behavior into a specific target experience. In this context Fullerton et. al. restate Katie Salen and Eric Zimmerman’s description of games as a “second-order design problem” (2004, 168). A designer designs the game, but the game delivers the experience; the designer does not create the experience directly.

This trend, to involve the player in the design process, is gaining momentum in both academia and development studios. The human-centered design or user-centered design, originating from software engineering, has been a big influence on this trend. It should come as no surprise that Microsoft’s game studios are front runners in this respect, as Microsoft has much experience with similar methods used in regular software development. Pagulayan et. al. (2003) describe the heuristics and structured user tests that have been used to develop several games within Microsoft. Slowly but surely these methods have become an integral part of designing games, and more and more these methods rely on a combination of qualitative methods such as heuristic evaluations (Sweetser & Wyeth, 2005; Schaffer, 2008) and quantitative metrics such as plotting the locations of player death’s onto a level map (Swain, 2008).

Play-centric design focuses on the process of designing games. By structuring the design process, involving the player, gathering data from prototypes, and iterating many, many times everything is done to ensure that the end product, the finished game, is as good as the design team can make it. For a professional game designer these methods are (or at least should be) regular tools of the trade. They do not make the process of designing games less hard, but they do help the designer to stay on track, break the task down into a series of smaller subtasks, and steadily progress towards a high quality end product.

With the proper methods and tools this process can be refined. The play-

centric approach would benefit from methods that can speed up each iteration or increase the improvements that are made in each one. There are several types of methods that seem applicable. Certainly formal models of game design are widely accepted amongst them, but also techniques to gather and process data collected during play-tests.

3.4 Game Vocabularies

Not only designing games is a hard task, talking about them is already difficult: there is no common language to describe their inner workings. There are plenty of books and articles that discuss games as rule-based systems, but almost all of these choose their own words. More often than not, these vocabularies are very good at describing particular games, but they rarely transcend into a more generic vocabulary.

In a 1999 *Gamasutra* article designer Doug Church sets out to create a framework for a common vocabulary for game design (Church, 1999). According to this framework, a game design vocabulary should consist of “formal abstract design tools”, where “formal” indicates that the vocabulary needs to be precise, and “abstract” indicates the vocabulary must transcend the particularities of a single game. For Church the vocabulary should function as a set of tools, where different tools are suited for different tasks, and not all tools are applicable for a particular game.

Doug Church describes three formal abstract design tools in his article:

1. **Intention:** Players should be able to make an implementable plan of their own creation in response to the current situation in the game world and their understanding of the game play options.
2. **Perceivable Consequence:** Game worlds need to react clearly to player actions; the consequences of a player’s action should be clear.
3. **Story:** Games might have a narrative thread, whether designer-driven or player-driven, that binds events together and drives the player forward toward completion of the game.

These three tools form a list that is by no means complete or exhaustive, nor did Doug Church intend it to be. Between 1999 and 2002 the *Gamasutra* website hosted a forum where people could discuss and expand the framework. The term ‘design tool’ was quickly replaced by the term ‘design lexicon’ indicating that the formal abstract design tools seem to be more successful as an analytical tool than a design tool. Bernd Kreimeier reports that “at least 25 terms were submitted by almost as many contributors” (Kreimeier, 2003). As a project the formal abstract design tools have been abandoned; however, Doug Church’s article is often credited as one of the earliest attempts to deal with the lack of a vocabulary for game design, even though his framework never caught on.

There are several researchers that carry on the torch that Doug Church lit. The “400 Project” initiated by Hal Barwood and Noah Falstein is one example (Falstein, 2002). Barwood and Falstein set the goal of finding and describing

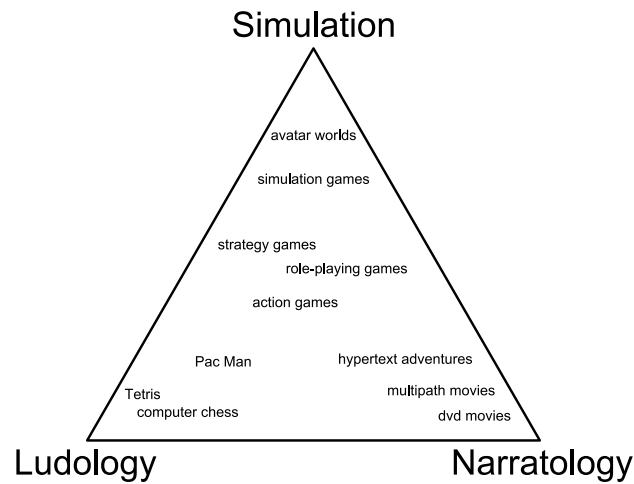


Figure 3.3: One of Craig Lindley’s taxonomies (after Lindley, 2003).

400 rules of game design that should lead to better games. The project website lists 112 rules.⁴ However, the project seems to be abandoned as well; the last update to the website was in 2006.

Craig Lindley (2003) uses orthogonal taxonomies to map games onto a hypothetical space defined by dominant game forms such as narratology (focus on story), ludology (focus on gameplay) and simulation (focus on realism). Individual games can be mapped to the space depending on their relative closeness to each of these extremes (see figure 3.3). Lindley describes a few possible, complementary taxonomies using one-, two- and three-dimensional spaces. He designed these taxonomies as a high level road map to inform the design team of the intended design target of a particular game project. The taxonomy also suggests suitable tools and techniques borrowed from other fields; a game that veers towards the narrative side will benefit more from traditional storytelling techniques than a game that is a simulation first and foremost. Lindley’s game taxonomies provide a systematic framework in which many of the formal abstract design tools can be embedded, providing structure to what otherwise would remain a loose collection of labels.

The game ontology project takes the notion of a common vocabulary for games into yet another direction. This project attempts to order snippets of game design wisdom into one large ontology. An ontology is a large classification scheme that has a hierarchical organization. Each entry in the ontology describes a common structure found in games. It lists strong and weak examples of the structure and lists parent and children categories. For example, the ontology entry “to own” is used to describe the game structure in which game entities can own other game entities. An example would be the game entity ‘Mario’ that collects ‘mushrooms’ and ‘stars’, etc. “To own” is a child of the

⁴http://www.theinspiracy.com/400_project.htm (last visited June 23, 2011).

“entity manipulation” entry which, in turn, has three children: “to capture”, “to possess” and “to exchange” (Zagal et al., 2005).⁵

The game ontology project aims to explore the design space of games without prescribing how to create good games. More than Doug Church’s formal abstract design tools, it primarily is an analytical tool; it aims at understanding games rather than building them. This is a general characteristic of this and other game vocabularies. Their success as an analytical tool does not translate easily to being successful as a design tool. Obviously, the development of a high level, consistent language to describe common game structures will help designers in the long run, and, as all the vocabulary builders point out, can be a great help in mapping the relatively unexplored areas of the game design. In fact, all authors describe how their vocabularies can be used as a brainstorming tool, simply by selecting and exploring random combinations of notions describing common aspects of games. However, no matter how useful this practice can be, it can usually only help with generating ideas. This is only a small part of the entire process of building a game, yet it requires considerable investment on the part of designers who must familiarize themselves with many new concepts to learn the vocabulary. The game ontology project, for example, consists of over one hundred separate entries, each of which ties in with several other entries in the ontology. For game developers it can be difficult to see what is the actual return on their investment in learning a vocabulary of that size.

The many different approaches towards a common vocabulary for games aggravate this problem.⁶ Every vocabulary has its own unique approach and terminology. Simply determining where and how all these approaches overlap or collide makes an extensive academic research project in itself. Even when a game designer invested the time and effort to learn one of these vocabularies, effectively working together or sharing knowledge with somebody who has learned a different vocabulary is still going to be a problem. The only thing all these vocabularies seem to share is their rejection by game designers. In the words of Daniel Cook: “Academic definitions of game design contain too many words and not enough obvious practical applications where people can actually use the proposed terminology” (2006).

3.5 Design Patterns

Staffan Björk and Jussi Holopainen’s work on game design patterns also seeks to address the lack of vocabulary for game design (2005, 4). However, their approach is slightly different as they drew inspiration from the design patterns found in architecture and urban design as explored in the works of Christopher Alexander. According to Alexander: “There is a central quality which is the root criterion of life and spirit in a man, a town, a building, or a wilderness.

⁵The project has an active web page where all entries can be found: <http://www.gameontology.com> (last visited July 8, 2011).

⁶The discussion here cannot address all vocabularies out there. One approach worth mentioning focuses on tracking gameplay innovations: the Game Innovation Database found at <http://www.gameinnovation.org/> (last visited October 22, 2010).

This quality is objective and precise but it cannot be named” (1979, ix). His pattern language is designed to capture this quality. Patterns are presented as problem and solution pairs, where each pattern presents a solution to a common design problem. These solutions are described as generically as possible so that they might be used many times (Alexander et al., 1977, x). The patterns are all described in the same format. Each pattern also has connections to ‘larger’ and ‘smaller’ patterns within the language, where smaller patterns help complete larger patterns (Alexander et al., 1977, xii).

This idea has been transferred to the domain of software design by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides.⁷ Within software engineering the principles of object-oriented programming take the place of Alexander’s unnamed quality. Software design patterns are a means to record experience in designing object-oriented software (Gamma et al., 1995, 2). Today, software design patterns are common tools in teaching and designing software.

A pattern framework for game design following these examples was suggested by Bernd Kreimeier (2002). However, Björk and Holopainen break away from existing design patterns. According to them, design patterns as problem-solution pairs do not suit game design because:

“First, defining patterns from problems creates a risk of viewing patterns as a methodology for only removing unwanted effects of a design rather than tools to support creative design work. Second, many of the patterns we identified described characteristics that more or less automatically guaranteed other characteristics in the game, in other words, the problem described in a pattern might easily be solved by applying a related and more specific pattern. Third, the effect of introducing, removing, or modifying a game design pattern in a game affected many different aspects of the gameplay, making game design patterns imprecise tools for solving problems mechanically. However, we believed that game design patterns offer a good model for how to structure knowledge about gameplay that could be used both for design and analysis of games.

Based on these conclusions, we have chosen to define game design patterns in the following fashion: game design patterns are semiformal interdependent descriptions of commonly reoccurring parts of the design of a game that concern gameplay.” (Björk & Holopainen, 2005, 34)

This decision makes their pattern approach indistinguishable from the game vocabularies discussed above, and subjects it to all the associated problems. Their book contains hundreds of patterns, and their website has hundreds more. This is indicative of Björk and Holopainen’s dedication to their framework, but also of the fact that their patterns are not built on a strong theoretical notion of what games are and how gameplay emerges from game parts. Their mention of games as state machines (Björk & Holopainen, 2005, 8) is not enough to carry

⁷Also known as the ‘Gang of Four’.

the weight of the whole framework. The number of patterns used by software engineering, by contrast, is much lower: a typical introduction has about twenty patterns. I doubt that the diversity of problems and solutions encountered in games is one order of magnitude larger than those encountered in software engineering. The real difference, is that software design patterns are based on the principles of object-oriented software design. This gives the patterns focus and provides leverage on the problems they need to deal with, leading to patterns that are further abstracted from typical applications or implementations. Without a clear theoretical vision on games, drafting patterns becomes an exercise in cataloging reoccurring parts of games, without ever questioning why they re-occur or whether these and related patterns might be the result of some deeper mechanism at work within games. Where Christopher Alexander starts from the notion that his design patterns ultimately allow us to approach some quality that cannot be named, but which is objective nonetheless, the game design patterns lack a similar theoretical focal point.⁸

Design patterns work well for architecture and software engineering because they codify a particular quality in their respective domain. In order to replicate their success for game design, a similar notion of quality within games should serve as its foundation. Unfortunately, Björk and Holopainen do not formulate such a quality for games. Without such a quality no set of game design patterns can be anything more than a vocabulary of games. The notion of games as state machines as mentioned by Björk and Holopainen could be the starting point to develop a notion of quality within games, an opportunity which is missed by Björk and Holopainen, but which I will explore further.

3.6 Mapping Game States

Games can be, and often are, understood as state machines: there is an initial state or condition and actions of the player (and often the game, too) can bring about new states until an end state is reached (see for example Järvinen, 2003; Grünvogel, 2005). In the case of many single-player video games either the player wins or the game ends prematurely. The game's state usually reflects the player's location, the location of other players, allies and enemies, and the current distribution of vital game resources. From a game's state the player's progress towards a goal can be read.

There are several techniques to represent state machines. Finite state machine diagrams, for example, represent state machines with a finite set of states and a finite set of transitions between states. One state is the initial state and there might be any number of end states. To represent a game as a finite state machine, all the states the game can be in need to be identified. Next all possible transitions from state to state need to be identified. For certain simple games this works. For example, figure 3.4 represents a finite state machine describing

⁸Although it must be noted that Alexander's pattern language also includes several hundred described patterns. In that sense game design patterns are not very dissimilar. However, Alexander's pattern language describes a fairly large number of domains: buildings, towns, etc. The sets that describe each individual domain are much smaller.

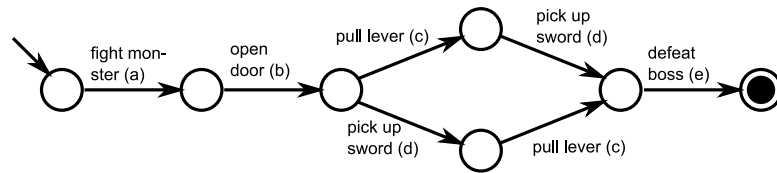


Figure 3.4: A finite state machine representing an adventure game.

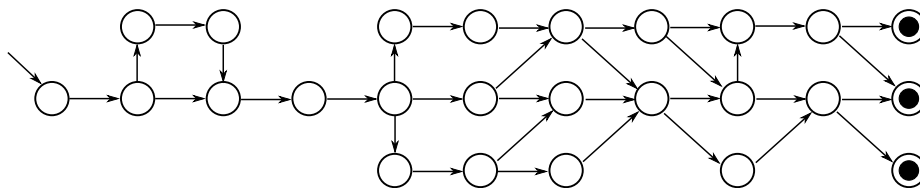


Figure 3.5: A more complex finite state machine, but one that still produces a finite set of trajectories.

a fairly straightforward and relatively simple, generic adventure game. This diagram utilizes a simplified version of the more elaborate state machines diagrams specified by Unified Modeling Language (UML) (Fowler, 2004, 107-116).

Many things have been omitted from this finite state machine. For example, the way the player moves through the world has been left out, which is no trivial aspect in an action-adventure game with a strong emphasis on exploring (as is the case with most action-adventure games). Still, movement is easily abstracted away from this diagram as movement does not seem to bring any relevant changes to the game state (other than the requirement of being in a certain location to be able to execute a particular action).

The question is whether or not this type of representation of a game is of any use. Looking at the diagram this game does not look complex at all. The possible set of different trajectories through the finite state machine is very limited. The only possibilities are *abcde* and *abdce*. This game is a machine that cannot produce any other result. It is, to use Jesper Juul's categories, a game of progression, and not a game of emergence (Juul, 2005, 5). To be fair, most adventure games have a much larger set of states and player actions that trigger state transitions. There might be side quests for the player to follow, or even optional paths that lack the symmetry of the two branches in figure 3.4. A game like this might grow in complexity very fast (see for example figure 3.5), but still the number of possible trajectories remains ultimately finite (unless one introduces loops, see below). Yet this is what many games have done in the past.

One way to create infinite possible trajectories is to introduce loops. Noam Chomsky has shown that by including loops in a state machine the set of possible results becomes infinite (Chomsky, 1957, 18-25). For example we could allow the player to go back after opening the door and fight another monster (see figure 3.6). The possible set of results is now $\{abcde, abdce, ababcde, ababdce,$

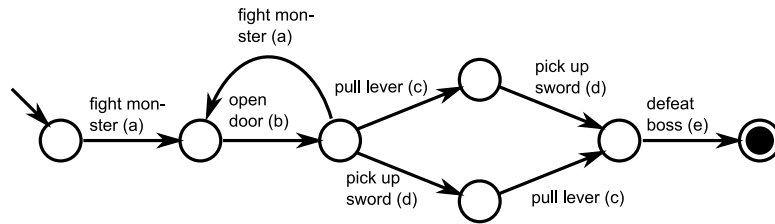


Figure 3.6: A looping finite state machine representing an adventure game.

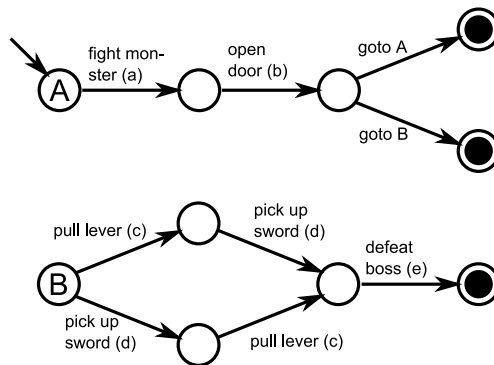


Figure 3.7: A recursive finite state machine representing an adventure game.

abababcde, abababdce, ..., etc.}; the player can fight an infinite host of monsters before proceeding. Of course, this has little purpose in the context of the game, unless the player is awarded experience points for defeating each monster and the number of experience points somehow affects the player’s chance of defeating the end boss (which might very well be the case in a computer role-playing game). However, if this were the case, one might argue that each level of experience would in fact constitute a new state, leading to an infinite number of states in the diagram.

Finite state machines lack a construct for memory, which would solve the experience points problem described above. To deal with the memory problem William A. Woods designed “augmented transition networks” which uses recursion and a stack (Woods, 1970). In an augmented transition network a transition might invoke a new state in the same or in a separate network of states and transitions. This would cause the old state to be pushed onto the stack and the new state to be activated. When the machine encounters an end state, it recalls the last state pushed on to the stack and proceeds from there. It only terminates when nothing is left on the stack. In an augmented transition network representation of the adventure game above, every time the player fights a monster the network would call itself recursively, and thus every fight would be pushed to the stack, and the number of fights on the stack can be checked when the player fights the boss monster (see figure 3.7).

Game states are usually much better expressed using a mix of variables and

states. Not only allows such a mixture to model the large number of states encountered in most games, it also shifts attention towards the transitions between the states, which corresponds to user actions. It is possible to construct a diagram for RISK with only four states and seven transitions (figure 3.8) in which each transition affects one or more variable registers representing territories, armies and cards. The diagram shows many loops, and as a result an infinite number of different paths through the state machine is possible. A diagram that focuses on transitions is clearly more capable to capture the nature of games and the varied sessions of play. However, the diagram is not very easy to read as some sort of pseudo code is needed to represent the exact mechanics that checks and changes the variable registers.

Petri nets are an alternative modeling technique suited for game machines that are explored by a few researchers (Natkin & Vega, 2003; Brom & Abonyi, 2006; Araújo & Roque, 2009). Petri nets work with a system of nodes and connections. A particular type of node (places), can hold a number of tokens. In a Petri net a place can never be connected directly to another place, instead a place must be connected to a transition, and a transition must be connected to a place. In a classic Petri net places are represented as open circles, transitions are represented as squares and tokens are represented as smaller, filled circles located on a place. In a Petri net tokens flow from place to place; the distribution of tokens over spaces represents the current state of the Petri net (see figure 3.9). This way the number of states a Petri net can express is much larger than with finite state machine diagrams. Petri nets put much more focus on the transitions and have a natural way of representing integer values through the distribution of tokens over the places in the network. Indeed, “Petri Nets tend to be, in general, a more economic representation when state-space complexity increases” (Araújo & Roque, 2009).

One of the very promising advantages of the use of Petri nets, is that they have a very solid mathematic foundation. Petri nets can be easily verified and simulated. They are almost like a visual programming language. But this ad-

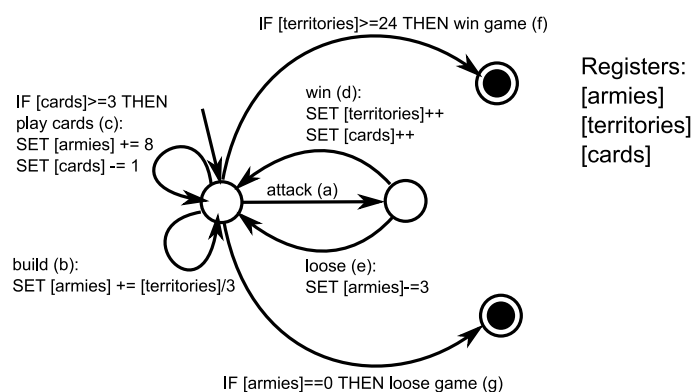


Figure 3.8: A state diagram for RISK

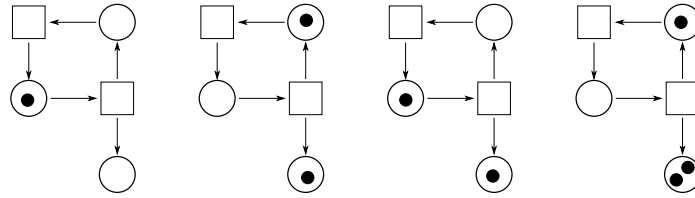


Figure 3.9: Four iterations of the same Petri net showing movement of tokens through the network

vantage often is a double edged sword. Petri nets can model a complete game with a high level of detail, but this frequently leads to quite complex diagrams which try to capture a game in its entirety. Petri nets can become the equivalent of a game's source code, and just as inaccessible to a non-programmer.

3.7 Game Diagrams

State machine diagrams and Petri nets are not the only diagrammatic approaches to deal with the problem of game design. Over the years, a few other diagrammatic or systematic approaches have been developed that deal with games exclusively. Game theory as invented by John von Neumann, can be seen as one of the earliest attempts to deal with game-like systems that feature a similar state-space explosion as we have seen with finite state machine diagrams. One could try to map this sort of systems with decision trees, but they would quickly grow out of control. Instead, game theory uses matrices to chart the gains or losses of possible moves in relation to the opponents move. From these matrices rational strategies, or the lack thereof, should become apparent (see Binmore, 2007). Emmanuel Guardiola and Stéphane Natkin (2005) use similar matrices to represent all possible interactions between a single player and a computer game. Game theory and its application in computer games focuses on the actions of the players. It is a very useful technique to balance actions and prevent dominant strategies to emerge. Game theory works best with relatively simple, two-player games; it seems to restrict itself mostly to a formal theory of gameplay decisions, which in itself is a relevant subset of game design. However, it does not scale very well to the scope of modern computer games, which includes much more elements (Salen & Zimmerman, 2004, 243).

Raph Koster's exploration in game diagrams presents yet another approach. Presented at the Game Developers Conference in 2005, his focus is on atomic particles that make up the game experience; on what he calls the 'ludemes' and devising a graphical language for them (Koster, 2005a). These 'ludemes' are essentially the core mechanics of a game. Koster proposes to harvest these ludemes by reverse engineering existing games. Sadly, as Koster points out himself, he does not succeed. Figure 3.10 shows his best take on diagramming CHECKERS. He believes games can be diagrammed, but he also admits that the language he came up with is not sufficient for the task.

Inspired by Raph Koster, Stéphane Bura takes the idea of creating game dia-

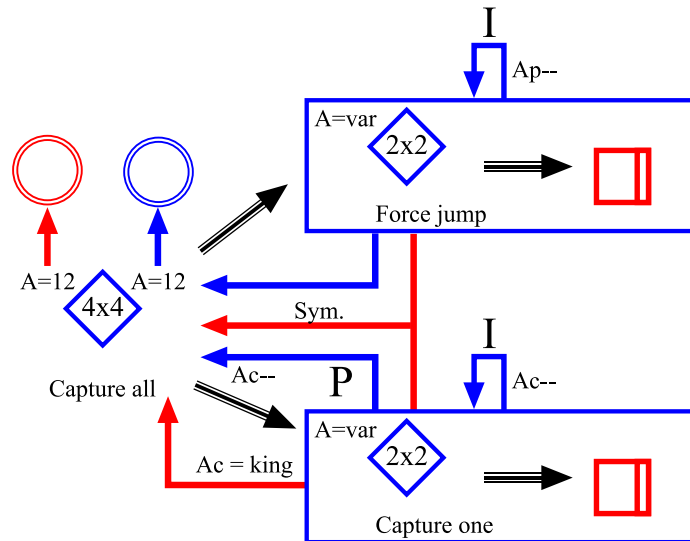


Figure 3.10: Raph Koster's diagram of CHECKERS (2005a).

grams one step further (2006). Combining Koster's approach with his experience with Petri nets, Bura designs game diagrams that try to capture a game's structure at a higher level of abstraction than simply its core rules. Removing game diagrams from the burden of modeling rules at the lowest level, allows Bura to focus more on the emergent properties of games. His diagram models notions such as 'skill' and 'luck' as abstract resources that affect other actions in the diagram, either by enabling or inhibiting them. Figure 3.11 shows the diagram Bura created to model BLACKJACK. As should become clear from this diagram, Bura tries to capture the entire 'gestalt' of the game into a single image. In this diagram the elements that model 'skill', 'luck' and 'money' are similar to places in a Petri net and can accumulate tokens. The elements 'gain' and 'risk' act like transitions. They consume and produce resources according to the arrows that connect them to other arrows. This diagram also includes two inhibiting connections (lines that end in a circle) to denote that the 'luck' of the house inhibits the 'gain' of the player and that the 'skill' of the player inhibits the money he or she risks. Although Bura is more optimistic than Koster, he also admits that much work still needs to be done. He suggests a standard library of ludemes to work with and sub-diagrams to increase the detail. But to my knowledge, none of these extensions have been published.

There are also a few examples of the use of UML for representing game systems diagrammatically. Taylor et al. (2006) extend UML use-case diagrams to describe game-flow: the experience of the player. Their focus is on the play session and the progression of the player through the game. Perdita Stevens and Rob Pooley use class diagrams, collaboration diagrams and state machines diagrams (three different types of UML diagrams) in their educational case study of modeling the structure of CHESS and TIC-TAC-TOE with standard UML

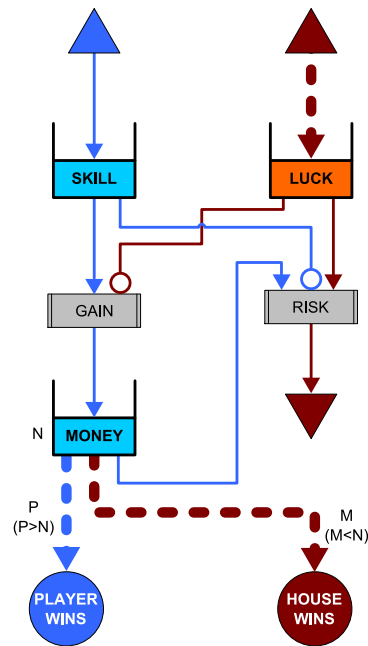


Figure 3.11: Stéphane Bura's diagram of BLACKJACK. (2006)

(Stevens & Booley, 1999, 178-189). These attempts suffer from problems similar to other types of game diagrams. As a specification language, UML can be very detailed and inaccessible to non-programmers. In a way, UML is too universal to capture the particular structures that are important in the domain of games.

3.8 Visualizing Game Economies

Working from an extended version of UML collaboration diagrams, as described by Bran Selic and Jim Rumbaugh (1998), I experimented with using UML notation to capture how different parts of a game interact (Dormans, 2008b). In contrast to the previous attempts to diagram game mechanics, I started from the idea that these diagrams should map the internal economy that drive the emergent behavior in many games as described by Adams & Rollings (2007). This perspective is discussed in further detail in Chapter 4. In these diagrams I focused on the flow of resources and flow of information between game elements. Figure 3.12 displays the UML diagram I created for the game POWER GRID. It shows the most important elements and the communication in the form of flow of resources and information between them. The solid lines and black shapes indicate flow of resources, and dotted lines and white connections indicate flow of information.

In the discussion of these diagrams I quickly zoomed in on the feedback structure that can be read from the diagram. Feedback is realized through a

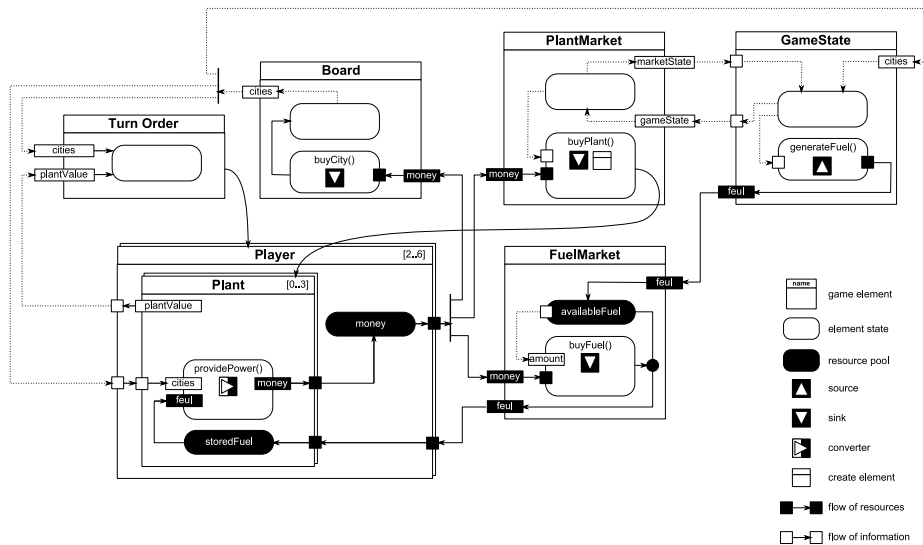


Figure 3.12: POWER GRID in an adaptation of UML collaboration diagrams.

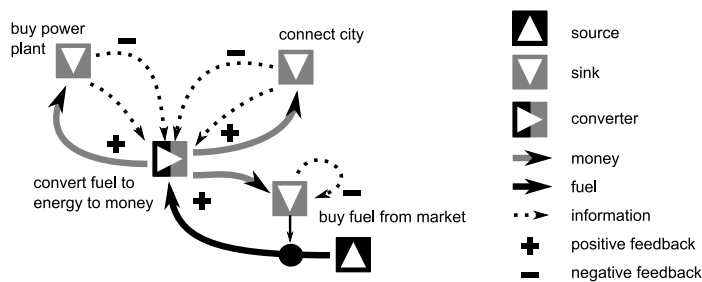


Figure 3.13: The feedback structure in POWER GRID.

closed circuit of communication. Figure 3.13 shows the feedback structure for POWER GRID in a shorthand notation that omits much of the UML. From this and other analyses I started extracting common patterns found in these feedback structures (see figure 3.14).

Although this approach produces interesting results and was positively received by game designers and researchers, there are a number of problems with it. As with many other approaches discussed in this chapter, it is geared towards documenting existing games, although the notation has been used as a brainstorming tool during design workshops. During brainstorming writing full UML by hand proved to be impracticable; the shorthand notation was used more often. Finally, the diagrams are static, while they represent a highly dynamic system. There are cues in the diagram that suggest the dynamic behavior of the game it represents, but it leaves too much room for interpretation: one person might read something completely different from the same diagram than the next

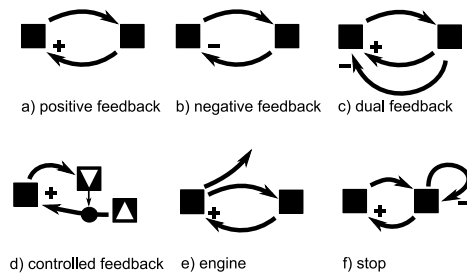


Figure 3.14: The feedback patterns extracted from POWER GRID.

person.

Independent from my work, Morgan McGuire and Odest Chadwicke Jenkins use a similar approach for their “commodity flow graphs” (see figure 3.15). These graphs also represent a game’s economy and flow of commodities (2009). Their approach is comparable to my attempts to map a game’s economy, but is different on three important points:

1. McGuire and Jenkins interpret the notion of economy quite literally and restrict themselves to games that include trading and tangible resources (“commodities”), whereas the notion of internal economy taken from Adams and Rollings ‘commodifies’ many more elements of games that are generally less tangible such as player and enemy health in a shooter game.
2. Although feedback does appear in one of the two examples, their graphs are not designed to foreground feedback mechanisms.
3. Their graphs are quite informal, even though some elements are explained in the accompanying text, many important details are not.

3.9 Industry Skepticism

Not everybody in the game industry thinks that developing design theory or methodology is a very good idea. In an interview with Brandon Sheffield, Raph Koster recalls that his presentation on game diagrams split his audience at the Game Developers Conference in 2005. Some thought it was good, some thought it was a complete waste of time (Sheffield, 2007). I have come across similar sentiments in discussions about my work with people working in the game industry. Usually those who dislike the premise of design methodology argue that they are academic toys with little relevance for real, applied game design; that they, indeed, are a waste of time. Another common argument against design methodology is that they can never capture the creative essence that is the heart of successful games. In this argument, design methodologies represent an attempt to destroy the very soul of the art of game design; no method can replace the creative genius of the individual designer (see Guttenberg, 2006). Starting with the first, I will address both arguments below.

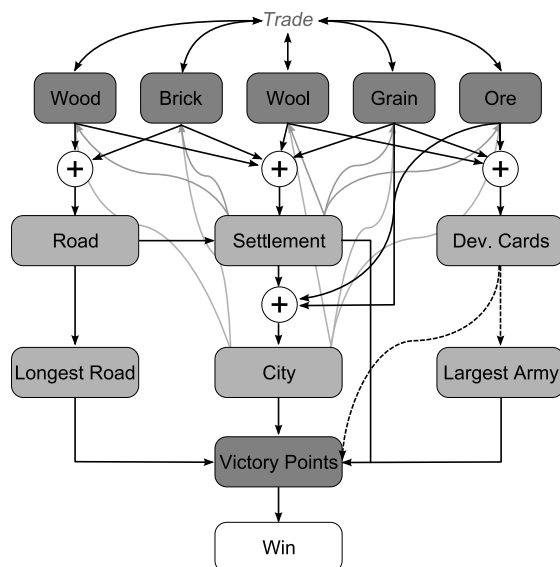


Figure 3.15: Commodity flow graph for SETTLERS OF CATAN, after McGuire & Jenkins (2009, 243).

The current vocabularies, aids and frameworks have a poor track record. As should have become clear from the discussion above, there are many of these out there, many of them designed by academics, not all of whom have actual, hands-on game design experience. The return value of using them, set against the often considerable investment required to learn them, is not particular high, especially for those tools that excel in analyzing games, which is done more often within universities than outside. The same goes for those frameworks that allow designers to explore the design-space. The design programs within universities allow for such exploration, whereas outside there is little time for such theoretical exploration. The argument that the industry too would benefit from such exploration is rather hollow if money needs to be made and one cannot afford to take chances on a new innovative concept.

The sentiment is valid, but does not cripple the effort to create game design methodologies. It simply suggests criteria for evaluating design methods: design methods should help design games, not just analyze them. This seems obvious, but many methods that have been developed over the years are analytical methods, even when they sometimes are presented as design tools. These methods help us understand existing games or explore the hypothetical design space, but offer little practical guidance on how to build them. What is more, design methods should return the investment required to learn them. The latter criterium can be met in two ways: make sure that the required investment is low, or make sure that the return is high. Obviously a design method should aim to do both in order to maximize the return on investment.

The second argument, that no design method can replace the creative genius

of the individual designer, is more problematic. People who subscribe to this opinion dismiss the whole idea of design methodology. However, this opinion is often informed by a rather naive conception of art. Art is, and always has been, the combination of creative talent, practiced technique and hard work. There is no point in denying that one artist has more talent than another, but pure talent rarely makes up for the other two aspects. Especially within an industry where much money rides on the success of each project, investors simply cannot afford to gamble on creative talent to deliver all the time.

The image of the artist as the creative genius is a romantic vision that rarely fits reality. To create art, one must learn the techniques of the trade and work hard. This has always been the case for all forms of art. There is no reason to assume that games are any different. The artist's techniques are many. They range from the practical to the theoretical. Painters learn how to use a brush with different types of paint, on the one hand, but learn about the mathematical principles of perspective and the psychological principles of cognition on the other. The development of abstract art throughout the nineteenth and early twentieth century has been a gradual and deliberate intellectual process (Rosenblum, 1975). The scientific invention of the perspective revolutionized Renaissance painting (Panofsky, 1960). The foundation of literary theory that Aristotle laid over two thousand years ago is still taught today (Vogler, 2007). What has changed over the years is the widening gap between artist and academic communities. During the Middle Ages art prevailed where academia hardly survived, as a result the artist and the academic frequently were the same person. These days, with thriving universities, being an academic has become a profession of its own, but that does not mean that the ties between art and academia have been severed. There are still many artists that contribute to the academic debate and there are still many academics that contribute to the evolution of art. Games are no different.

In contrast to what skeptics of design methodologies fear, design methods help shape games but they cannot replace the creative genius. No matter how good a method or tool is, it can never replace the vision of the designer, nor can it replace the hard work involved in designing a game. At best it can ease the burden and refine one's techniques. Sometimes methods and tools seem restrictive; when holding a hammer everything starts to look like a nail. But the best methods do not restrict a designer's vision. Rather, they should enhance it, enabling them to work faster and create better results. Ideally, design methods also facilitate teamwork and collaboration. For example a design tool that allows accurate representation of game elements, would reduce the chance that individual team members end up working toward different visions.

Game designers that take no interest in design methodology are either foolish or lazy. However, designers have all rights to be critical of design methods, and I do hope they remain so in the future. After all, they are the final judges that decide whether or not a given method is worth their time and ultimately expand their expressive power with the medium of games.

3.10 Conclusions

To summarize, over the years a number of frameworks, vocabularies and work methods have been created to assist game designers, with varying success. Game design documents are generally considered cumbersome and inefficient; they are seldomly put to good use. Everybody uses game design documents in their own way. For some designers, these documents capture the creative direction early in the development process, while for others they are a tedious requirement of the job of game designer. For the purpose of the discussion here, no generic wisdom to aid the development of design tools can be extracted from the diffuse practice of writing design documents.

The MDA framework provides a useful lens on the different aspects of game design, and can help designers to understand where to start the huge task of designing a game. It breaks down games into three understandable and useful layers, and teaches inexperienced designers to look through the outer layers of a game into the *mechanics* core. However, the framework has evolved little over the years, and close examination of its core concepts is likely to raise more questions than it answers. To serve as a design tool that goes beyond the very basics of game design; the MDA framework lacks scrutiny and accuracy.

Player-centric design practices, where short iterations and frequent play-testing are the key, are more successful in structuring the hard and laborious process of designing games. However, there is room for improvement. With the proper methods and tools every iteration can be made more effective, and new ways of gathering qualitative and quantitative data might present themselves. The theory and tools presented in this dissertation are best embedded within in play-centric design process: they can help designers to improve every iteration, but they cannot take away the necessity to build prototypes and test them with real players.

There have been a number of attempts to create game vocabularies and pattern libraries that allow us to talk about games in better, more accurate terms. However, none of these vocabularies has really gained enough momentum to become something resembling a standard that spans both industry and academia. From a pragmatic point of view, these vocabularies require a considerable effort to learn while they are most successful in the analysis of existing games; they seem to be more useful for academics than they are for developers. In addition, they usually lack a clear theoretical vision on the artifacts they intend to describe. The result is that these vocabularies hardly scratch the surface of games and fail to contribute much to what most designers already knew intuitively.

The use of finite state machine diagrams or Petri nets to map games as state machines are both valuable techniques with a proven track record in their respective domains. However, their respective difficulties in capturing the essence of games indicates that simply framing games as state machines is not good enough. The number of game states usually is not finite, and their complexity quickly becomes problematic if one tries to model a game in every detail. A theoretic perspective on games first needs to develop a concise and objective notion of quality in games before it can help us understand their inner machinations

from a more generic scope. Once this notion has been developed, a diagrammatic language can be devised to represent these machinations. Petri nets are more promising but are less accessible to designers.

Game specific diagrams are a relatively unexplored approach towards the development of game design tools. Apart from some preliminary attempts by Koster, Bura, McGuire & Jenkins, and myself, little is done in this area. None of these attempts can claim to be successful and accepted by the game development community. Yet, the results are interesting, especially if they focus on a more abstract gestalt of games. A more abstract and generic scope to represent game designs seems to come quite natural to diagrams. At the same time, they are fairly easy and intuitive to learn: most diagrammatic languages utilize only a few core, reusable elements. When these elements express a generic and objective notion of quality in games, these diagrams could become quite powerful.

Game design tools are needed; they can be used to improve the process of game design, but the poor track record of current academic approaches created some resistance within the game industry against the whole notion of design methodology. Part of this resistance is understandable, as methods frequently fail to return the investment required to use them. This means that we need to rethink how design methods and tools should be used: they should not only facilitate analysis or theoretic exploration of game concepts, rather they should really help the designer to design. We should also take note of the fact that game design methods cannot replace the creative talent of the individual game designer. Game design methods should refine a designer's technique and increase the designer's expressive power, any game design method should ultimately be a tool, but it remains up to designer to make those tools work.

The object in constructing a dynamic model is to find unchanging laws that generate the changing configurations. These laws correspond roughly to the rules of a game.

John H. Holland (1998, 45)

4

Machinations

In the previous chapter we have seen that there are no consistent and widely accepted methodologies available for the development of games. Yet, the number of attempts and calls for such endeavors, indicate that a more formal approach to game design is warranted. In this chapter I propose a formal framework that focuses on discrete game mechanics called ‘Machinations’. Initial steps towards the current framework were presented at the Meaningful Play Conference in 2008 and the GAME ON-NA Conference in 2009 (Dormans, 2008b, 2009). The last sections appeared earlier in a paper presented at the Workshop on Artificial Intelligence in the Design Process at the AIIDE Conference in 2011 (Dormans, 2011d).

Taking into account the discussion of existing methodologies and the nature of games in the previous chapters, the Machinations framework presented here adheres to the following requirements:

1. A formal framework for game mechanics should formulate a clear theoretical vision on the structure of game mechanics and their relation to quality in games.
2. A formal framework for game mechanics should not only be an analytical tool for existing games; rather it should have a direct application for the development of new games.
3. A formal framework for game mechanics should provide a good return on the investment to learn it, either by keeping the investment low, or by making the return high, but preferably both.

The first section presents an overview of the Machinations framework and its various components. Sections 4.2 through 4.5 describe the Machinations diagrams that are part of the framework and how they can be used to represent game mechanics. The sections 4.6 to 4.8 describe how, using Machinations diagrams, feedback structures, and recurrent patterns in these structures, can be analyzed and related to a game’s dynamic behavior. In sections 4.9 and 4.10, I will discuss how the Machinations software tool, developed as part of this research, implements Machinations diagrams, and how it can be used to simulate, and generate quantitative data in order to balance games. The chapter

concludes with a case study. The Machinations framework is used to explore the theoretical game SIMWAR described by Will Wright, which, to my knowledge, has never been implemented. With this example I hope to illustrate that the Machinations framework can be used to simulate and analyze games even before they are built.

A word of caution: the Machinations framework is a lot to take in at once. The framework comprises many interrelated concepts that are best understood in unison. This means that there is no real natural starting point to explain all these concepts. I advise the reader to occasionally refer back to earlier concepts. I also like to point out that appendix A presents a single page overview of the most important concepts. In addition, many of the diagram examples can be found online in an interactive version at the Machinations wiki page: www.jorisdormans.nl/machinations/wiki.

4.1 The Machinations Framework

The Machinations framework formalizes a particular view on games as rule-based, dynamic systems. It focuses on game mechanics and the relation of these mechanics and the dynamic gameplay that emerges from them. It is based on the theoretical notion that structural features of game mechanics are for a large part responsible for the dynamic gameplay of the game as a whole. Game mechanics and their structural features are not immediately visible in most games. Some mechanics might be close to a game's surface, but many are obscured by the game's system. Only a detailed study of a game's mechanics can shed a light on the game's structure. Unfortunately, the models that are used to represent game mechanics, such as representations in code, finite state diagrams or Petri nets, are complex and not really accessible for designers. What is more, they are ill-suited to represent games at a sufficient level of abstraction, on which structural features, such as feedback loops, become immediately apparent. To this end, the Machinations framework includes *Machinations diagrams* which are designed to represent game mechanics in a way that is accessible, yet retains the structural features and dynamic behavior of the games they represent.

The theoretical vision that drives the Machinations framework is that gameplay is ultimately determined by the flow of tangible and abstract resources through the game system. Machinations diagrams represent these flows and foreground the feedback structures that might exist within the game system. It is these feedback structures that for a large part determine the dynamic behavior of games. This is consistent with findings in the science of complexity that studies dynamic and emergent behavior in a wide variety of complex systems (see section 1.5). By using Machinations diagrams a designer can give game systems a shape which would normally remain invisible. The main premise is that through Machinations diagrams, structure and quality in game mechanics become tangible.

Machinations diagrams are designed to capture game mechanics. As such, they are not only a design tool; they are also useful as an analytical tool to compare and analyze existing games. The Machinations framework allows us to

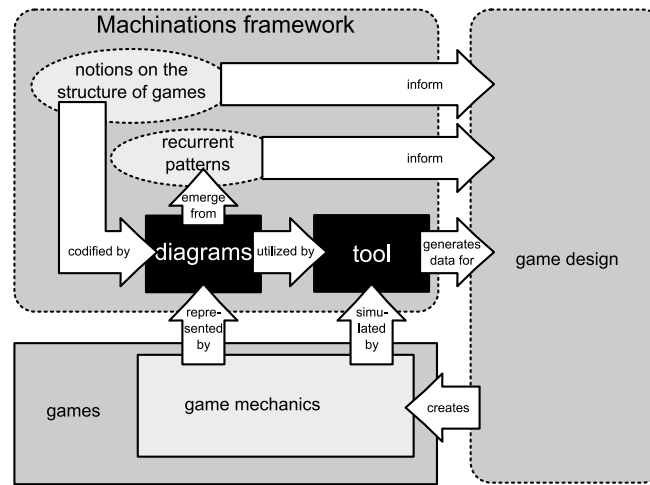


Figure 4.1: A map of the Machinations framework.

observe recurrent patterns across many different games. Machinations diagrams are a medium to express and reason about these patterns.

Machinations diagrams can be drawn with any tool. The language was designed to draw easily on a computer, or on paper. At the same time, the syntax of the language is exact. It describes unambiguously how different elements interact. This allowed the development of a *Machinations software tool*, which can be used to simulate and experiment with game systems. With this tool, a user can run and interact with a Machinations diagram. To a certain extent, a user can play a game represented as a Machinations diagram. In addition, the Machinations tool allows users to define ‘artificial players’ that interact with a diagram automatically. This means that games can be simulated without any actual interaction of real users. Such a simulation can run very quickly, allowing a user to experiment and gather quantitative data from thousands of simulated play sessions quickly and efficiently. To support this, the tool features automatic charts to collect data from each simulated play session.

Figure 4.1 is an overview of the Machinations framework and its most important components. It summarizes the discussion above.

Machinations diagrams create an abstracted perspective on game mechanics and are often used to focus on certain aspects of a game. The framework does not dictate how much detail a diagram should capture or what aspects of the game economy one should represent. Using Machinations diagrams many different aspects can be captured at many different levels of detail. The best perspective and level of abstraction is largely determined by context and purpose. Often it is sufficient to model games from the perspective of a single player, even if the game is actually played by multiple players. In these cases, it is often fairly easy to imagine how a diagram might be duplicated and combined to represent the multiplayer situation. In other cases it is useful to model one player at a

higher level of detail than other players. Likewise, particular aspects of games such as taking turns might be abstracted away. At a high level of abstraction, there often is little difference in the effects of players acting simultaneously and asynchronously, or in alternating turns. I have tried to keep the level of detail low and the level of abstraction high in the examples used in this chapter. This way the structural features of the internal economy are best foregrounded, which best serves the purpose of examining the effects of these structures on emergent gameplay. For this reason the natural scope for Machinations diagram is that of a single player and his or her individual perspective on the game system. Although it is certainly possible to model multiplayer systems, the framework, as it currently stands, does not include features designed to support multiplayer games in particular. For example, the main input device for interaction with a Machinations diagram is the mouse; there is no support for multiple input devices. Likewise, turn taking is geared towards a single player only: the system responds to every turn in a similar way. There is no support for alternating turns for a number of players; and it does not prevent the players to take actions outside ‘their turn’.¹

The Machinations framework focuses on rules and mechanics and does not take into account all elements of game design. Most importantly, the Machinations framework excludes all elements of level design. As such, the framework is more effective for games that do not rely on level design that much. This includes most board games, strategy games and management simulation games. While the framework will still be applicable to games that rely more on level design, for these games the framework can only describe a part of the picture. Level design will be the subject of the next chapter.

Moreover, The Machinations framework does not involve the player or any cultural dimension of representation through games. The framework treats games as complex state machines: interactive devices that can be in many different states, and whose current state determines the transition to a new state. This is an approach that can be and has been critiqued: without players games would be, quite literally, meaningless. The formal rule systems of games are subject to constant change and reinterpretation by players. A formal approach always runs the risk of turning a blind eye towards this dynamic and important dimension of games (see Malaby, 2007). However, building game systems is an important task of a game designer. It is this system that codifies the player’s possible interaction and generates individual game experiences. The aim of the Machinations framework is to understand the elementary structures that contribute to emergent gameplay and that ultimately facilitates the expressive and dynamic nature of games.

Finally, one more word of caution: when one sets out to model anything as complex as games, a model can never do justice to the true complexity of the reality of gameplay. The best models succeed in stripping down the complexity of the original by leaving out, or abstracting away, many important details. This is certainly the case with the framework and diagrams I present here.

¹This type of mechanics can be modeled using the current framework, but it does require fairly elaborate constructions. There is no ‘native’ support for this type of features.

However, any model is a tool that can help us understand and work with complex systems. To be able to use the model to the best effect, understanding the concepts that informed the creation of the model is required. As any model, the Machinations framework and diagrams only facilitate understanding; they are never a substitute for it.

4.2 Flow of Resources

The Machinations framework utilizes the idea of ‘internal economy’ (Adams & Rollings, 2007) to model activity, interaction and communication between game parts within the game system.² A game’s economic system is dominated by the flow of resources. In games resources can be anything: from money and property in *MONOPOLY*, via ammo and health in a first person shooter game, to experience points and equipment in a role-playing game. More abstract aspects of games, such as player skill level and strategic position, can also be modeled through the use of resources. A game’s internal economy consists of these resources as well as the entities or actions that cause them to be produced, consumed and exchanged. In the case of *RISK*, territories, armies and cards are the main resources; the player’s option to build will produce more armies, while with an attack the player risks armies to gain territories and cards.

In order to model a game’s internal economy Machinations diagrams uses several types of nodes, that pull, push, gather and distribute resources. Resource connections determine how resources move between elements and while state connections determine how the current distribution of resources modifies other elements in the diagram. Together, these elements form the essential core of Machinations diagrams. This section explains the basics of resource flow. The next section discusses different types of state connections. Sections 4.4 and 4.5 discuss more specialized nodes used to represent common operations in the game economy. An two-page visual overview of these core elements can be found in appendix A.

Machinations diagrams represent the flow of resources between different game elements. In this respect, Machinations diagrams are dynamic: just like tokens in a Petri net can move between places, resources in a Machinations diagram can move between nodes. The flow of resources is dictated by resource connections

²The history of the notion of internal economy within games is somewhat fragmented. It appears in a chapter title in Rollings & Adams (2003) which is a precursor to Adams & Rollings (2007), but the notion itself is not really discussed in that book. There are a few examples of the use of the term or the synonym ‘in-game economy’ in the context of games, such as in Simpson (2000), Burke (2005) and McGuire & Jenkins (2009). In these three cases the term is reserved for trade and inventory systems (as opposed to, for example, combat systems). McGuire and Jenkins even include ‘commodity flow graphs’ that visualize the flow of resources through a game, but they do not foreground feedback structures as Machinations graphs do (see discussion of their work in section 3.7). It is not until the notion was discussed in more detail in Adams & Rollings (2007) that it started to encompass more types of resources (such as health) than a strict interpretation of ‘economic’ would allow, and that a game’s internal economy could actually include combat systems, leveling systems, etc., as well. Since then, the term appears in lectures and syllabuses that follow Adams and Rollings book. It is in this wider sense that the term is used here.

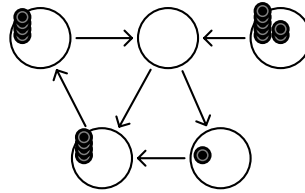


Figure 4.2: In a Machinations diagram the distribution of resources over nodes represents the state of a game. Resource connections indicate how resources might be redistributed.

represented as solid arrows connecting the nodes of the diagram. The state of the game corresponds to the distribution of resources over the nodes (see figure 4.2). Thus on one hand, a Machinations diagram represents a single state a game can be in, but on the other it determines subsequent states that are its direct result. The nodes and connections of the diagram visualize the structure that ultimately shapes the probability space of game states. The static version of Machinations diagrams accompanying this text obviously cannot display the state changes over time. The distribution of resources in a static Machinations diagrams always represent a game's initial state, unless stated otherwise.

In a Machinations diagram the nodes are active: they can *fire* and by firing they cause resources to be redistributed. A node in a Machinations diagram can be in one of three different activation modes:

1. A node can fire *automatically*: it fires at intervals determined by the diagram's time mode (see below). All automatic nodes fire simultaneously.
2. A node can be *interactive*, which means that it represents a player action and fires in response to that action. In the dynamic version of a Machination diagram, interactive nodes fire after users click on them.
3. A node can be *passive*, which means it can only fire in response to a trigger generated by another element (see section 4.3 below). Passive nodes still accumulate resources.

As Machinations diagrams are dynamic it is important to understand how they handle time. There are three different time modes for Machinations diagrams:

1. In *synchronous time mode* all automatic nodes are activated at regular intervals of arbitrary length specified by the user. All interactive nodes clicked by players fire at the next time step, at the same time when automatic nodes fire. In this mode all actions in one time step take place simultaneously. It is possible to for a user to activate multiple interactive nodes during a time step, but during a time step, each interactive node can only be activated once.

2. In *asynchronous time mode* automatic nodes in the diagram are still activated at regular intervals of arbitrary length specified by the user. However, players can activate interactive nodes at any time within the intervals and the resulting actions are executed immediately. In this case an interactive node can be activated multiple times during a time step.
3. Alternatively, a Machinations diagram can be in *turn-based mode*. In this mode time steps do not occur at regular intervals. Instead a new time step occurs after the player has executed a specified number of actions. This is implemented by assigning a number of action points to each interactive node and allotting players a fixed budget of action points each turn. After all action points are used, all automatic nodes fire and a new turn starts.³

The flow of resources from node to node is instantaneous: resources simply disappear and reappear at their new location. However, in order to help designers understand how the internal economy works, the Machinations tools can be set to visualize the resource flow by animating the movement of the resources along the resource connections. This is a visual aid only and does not affect the diagram in any way.

Resource connections, that dictate how resources are distributed when nodes fire, have a *label* that determines the flow rate. Many labels are numbers indicating a fixed flow rate, but the Machinations framework also uses simple expressions or icons to represent flow rates based on randomness, skill or other uncertain factors outside the game mechanics. When the label is omitted, the flow rate of a resource connection is considered to be 1.

The most basic nodes connected by resource connections are pools. They are represented as open circles. A pool collects resources that flow into it. Pools can be used to represent any collection of resources in a game. For example, the money possessed by a player of MONOPOLY can be represented as a pool, the money in the 'bank' can be represented as another pool. The resources themselves are represented as small dots. Different colors can be used to denote different types of resources. Alternatively, to avoid visual clutter, the number of resources on a pool can be represented as a number. Resource connections that lead into a node are considered to be its inputs while resource connection that lead out of a node are considered to be its outputs.

The activation mode of a pool is indicated by its visual appearance. Passive pools have a single outline, interactive pools have a double outline, automatic pools have a single outline and are marked with a star (*, see figure 4.4). When a pool fires it will try to pull resources through its inputs. The number of resources pulled is determined by the rate of the individual input resource connection. Alternatively, a pool can be set in 'push mode'. In this mode, a pool will push resources along its outputs when fired. Again, the number of resources pushed is determined by the flow rate of the output resource connection. A pool

³It is possible to create interactive nodes that cost no action points to activate. When all interactive nodes cost no action points, except a single 'end turn' action (that has no other effect), this can be used to create a game where players can take any number of actions until they are done.

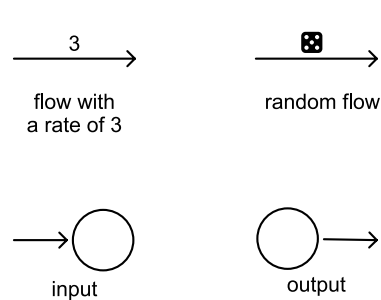


Figure 4.3: resource connections in a Machinations diagram.

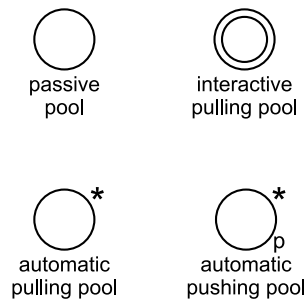


Figure 4.4: Different activation modes of a pool.

in push mode is marked with a 'p' (see figure 4.4). A pool that has only outputs is always considered to be in push mode, in which case the marker is omitted.

It might happen that two pools try to pull resources from the same source simultaneously, while there are not enough resources to serve both pools. For example, in figure 4.5 every time step pool B automatically pulls one resource from A and both C and D attempt to pull one resource from B. This means that after one time step, B will have one resource and C and D will both try to pull it. How this is resolved depends on the the time mode. In synchronous time mode, neither C nor D can pull the resource. After two iterations when B has pulled a second resource, both C and D will pull one resource from B. While the diagram runs, C and D will both pull a resource once every two time steps simultaneously. As A starts with nine resources, after nine time steps C and D will have four resources and one resource will remain on B. The state of the diagram will then no longer change.

In asynchronous or turn-based mode, either C or D will pull a resource one. Which pool has priority is initially random; subsequently, the priority alternates every time step. This means that C and D will both pull one resource from B on alternating time steps and eventually there will be four resources on C and five on D, or vice versa.

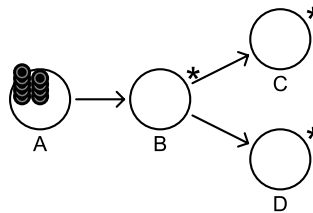


Figure 4.5: How simultaneous pulls are handled in a Machinations diagram depends on the diagram's time mode.

4.3 Flow of Information

Machinations diagrams are dynamic beyond the flow of resources. Resources flow rates can be modified by changes in resource distribution. In addition, nodes can be triggered, activated or deactivated in response to changes in resource distribution. These modifications are indicated by a second class of edges called *state connections*. State connections indicate how the current state of a node, the number of resources on it, affects target elements in the diagram. State connections are represented as dotted arrows. Labels and the type of elements they connect specify the type of state connection. There are four types of state connections that are characterized by the type of elements they connect and their labels:

1. *Label modifiers* connect a source node to a target label (L) of a resource connection or a state connection. It indicates how state changes of the source node (ΔS) modify the value of the target label as indicated by the modifier's label (M). The new value takes effect on the subsequent time step. Thus, the next value of label that is the target of a number (n) of label modifiers is given by the following formula:

$$L_{t+1} = L_t + \sum_{i=1}^n (M_i \Delta S_i)$$

For example, in figure 4.6 every resource added to pool A adds 2 to value of the resource flow between pools B and C. Thus the first time B is activated, 1 resource flows to A and 3 resources flow to C, the second time, 1 resource still flows to A, but now 5 resources flow to C. The label of a label modifier always starts with a plus or minus symbol indicating incrementation or decrementation.

Label modifiers are frequently used to model different aspects of game behavior. For example, a pool might be used to represent a player's accumulated property in a game of MONOPOLY. The more property a player has, the more likely it is that player will collect money from other players. This can be represented by the diagram in figure 4.7.⁴ Note that in this

⁴Note that many details are omitted in this diagram, in particular the diagram does not

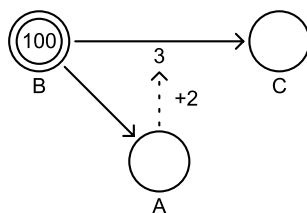


Figure 4.6: A label modifier affecting the flow rate between two pools. In effect: $Flow_{BC} = 3 + 2A$

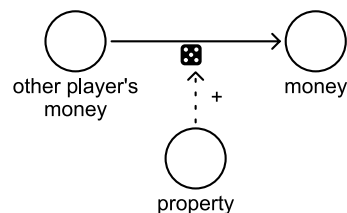


Figure 4.7: In MONOPOLY the state of your property positively affects the chance other player's money flows to you.

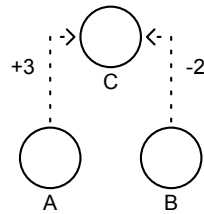


Figure 4.8: Node modifiers affect the number of resources on a pool. In effect: $C = 3A - 2B$

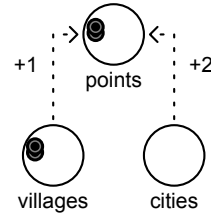


Figure 4.9: In *SETTLERS OF CATAN* your score is determined by the number of villages and cities you possess (amongst other things).

case the exact value of the label modifier is unspecified, it only indicates that the effect on the random flow rate is positive.

2. *Node modifiers* connect two nodes. They indicate how state changes of the source node (ΔS) modify the number of resources on the target node (N) as indicated by the modifier's label (M). The state changes to the target node are further processed during the subsequent time step. Thus, the new number of resources on a node that is the target of a number (n) of node modifiers is given by the following formula:

$$N_{t+1} = N_t + \sum_{i=1}^n (M_i \Delta S_i)$$

Figure 4.8 illustrates a node with two modifiers. By using negative node modifiers or redistributing resources from a node that has positive input node modifiers it becomes possible that the number of resources on a node becomes negative. In this case, the negative number of resources indicates a shortage. No resources can be pulled from a node that has a shortage, and resources that flow into a node with a shortage are used to compensate for the shortage first.

Node modifier can have labels that are fractions, for example '+1/3' or '-2/4'. In this case the number of resources of a target node is modified by the value indicated by the fraction's numerator every time there is a change to the number resources on the source divided by the fraction's denominator and rounded down. Thus when a source node changes from 7 to 8, the number of resources on the target is lowered by 2 if the modifier is -2/4, but if the modifier is +1/3 the number of resources on the target node does not change.

A simple, real-life example of the use of node modifiers can be found in *SETTLERS OF CATAN* where players gain 1 point for every village in their possession and 2 points for every city in their possession (see figure 4.9).

show how a player might acquire property. Diagrams representing *MONOPOLY* that paint a more complete picture are included later in this chapter.

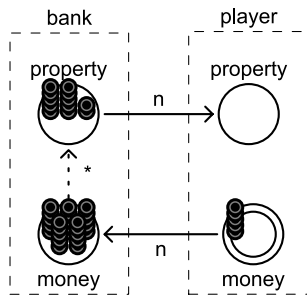


Figure 4.10: A trigger in MONOPOLY enables the acquisition of property by spending money.

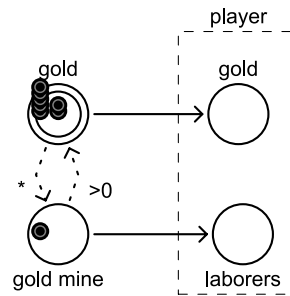


Figure 4.11: In CAYLUS the presence of a laborer at a goldmine activates the option to collect gold.

Node modifiers change the number of resources on other nodes. In effect, the use of node modifiers causes production or consumption of resources. These effects can also be achieved with a slightly more elaborate, but also somewhat cumbersome construction. This construction includes other elements that are introduced later. However, as the use of node modifiers is quite common, they are useful ‘syntactic sugar’: simpler notations of more elaborate constructions. For this reason node modifiers are here treated as a particular type of state modifiers.

3. *Triggers* are state connections that connect two nodes or connects a source node to the label of a resource connection. Triggers fire when all the inputs of its source node become *satisfied*: when each input passed the number of resources to the node as indicated by its flow rate.⁵ A firing trigger will in turn fire its target. When the target is a resource connection, it will pull resources as indicated by its flow rate. A node that has no inputs will fire outgoing triggers whenever it fires (either automatically, or in response to a player action, or to another trigger). Triggers are identified by their label which is a star (*).

Triggers are commonly used in games to react to redistribution of resources. For example, in MONOPOLY players might transfer money to the bank in order to ‘trigger’ the transfer of property from the bank into their possession. This can be represented as the diagram in figure 4.10.

4. *Activators* connect two nodes. They activate or inhibit their target node based on the state of their source node and a specific condition. The activator’s label specifies this condition. Conditions are written as an arithmetic expression (for example ‘=0’, ‘<3’, ‘>=4’ or ‘!=2’) or a range of values (for example ‘3-6’). If the state of the source node meets this condition then the target node is activated (it can fire). When the condition is not met the target node is inhibited (it cannot fire).

⁵This means that resource connections have a memory, they keep track of how many resources were requested and how many were delivered to pulling resources.

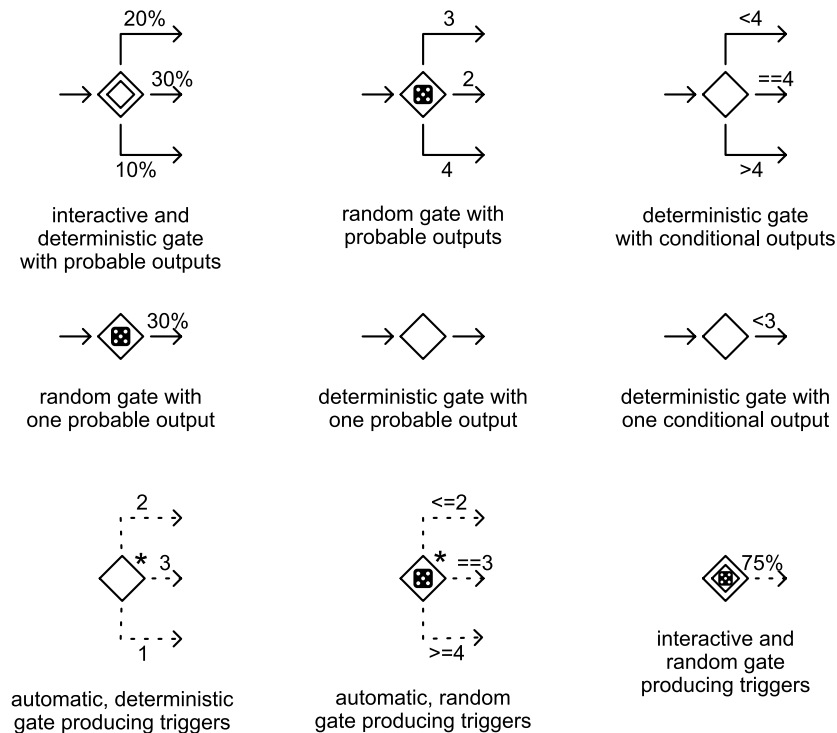


Figure 4.12: Different types of gates in a Machinations diagram.

Activators are used to model many different game mechanics. For example, in the board game CAYLUS players place their laborers (a resource) at particular buildings on the board to enable them to execute special actions associated with that building. For example, a player might place a laborer at a goldmine in order to collect gold (see figure 4.11). However, as indicated by the trigger in figure 4.11, in CAYLUS every time a player exercise this option, the laborer is returned to the player's laborer pool.

4.4 Controlling Resource Flow

Pools are not the only possible nodes in a Machinations diagram. Gates are another type of node. In contrast to a pool a gate does not collect resources, instead it immediately redistributes them. Gates are represented as diamond shapes that often have multiple outputs (see figure 4.12). Instead of a flow rate, each output is labeled with a probability or a condition. The first type of outputs are referred to as *probable outputs* while the other are referred to as *conditional outputs*. All outputs of a single gate must be of the same type: when one output is probable, all must be probable and when one output is conditional, all must be conditional.

Probabilities can be represented as percentages (for example ‘20%’) or weights indicated by single numbers (for example ‘1’ or ‘3’). In the first case a resource flowing into a gate will have a probability equal to the percentage indicated by each output, the sum of these probabilities should not add up to more than 100 percent. If the total is less than 100 percent there is a chance that the resource will not be sent along any output and is destroyed. In the latter case the chance that a resource will flow through a particular output is equal to the weight of that output divided by the sum of the weights of all outputs of the gate. Gates with probable outputs can be used to represent chances and risks. For example, in RISK players risk armies in order to gain territories. This type of risk can easily be represented by a gate with probable outputs indicating the rates for success or failure.

An output is conditional when it is labeled with a condition (such as ‘>3’ or ‘==0’ or ‘3-5’). In this case, all conditions are checked every time a resource arrives at the gate and one resource is sent along every output whose condition is met. As the conditions might overlap this can lead to duplication of resources, or, when no condition is met, to the destruction of the resource.

Like pools, gates have three activation modes: gates can be passive, interactive or automatic. Interactive gates also have a double outline and automatic gates are also marked with a star. When a gate has no inputs, it triggers every time it fires, this way gates can be used to produce triggers either automatically or in response to player actions.

Furthermore, gates have one of two different distribution modes: deterministic distribution and random distribution. A deterministic gate will distribute resources evenly according to the distribution probabilities indicated by percentages or weights if it has probable outputs. When it has conditional outputs it will count the number of resources that have passed through it every time step and uses that number to check the conditions of its outputs.⁶ A deterministic gate has no special symbol and is represented as a small open diamond.

A random gate generates a random value to determine where it will distribute incoming resources. When it has probable outputs it will generate a suitable number (either a value between 0 and 100 percent, or a number below the total weights of the outputs). When its outputs are conditional it will produce a value between 1 and 6 to check against the conditions, just as if the diagram rolled a normal six-sided die.⁷ Random gates are marked with a dice symbol.

Gates might have only one output. Gates with one output act exactly the same way as gates with multiple outputs. The gates on the middle row of figure 4.12 will (from left to right) randomly let 30 percent of all the resources pass, immediately pass the resource to the output regardless of the output’s flow rate, and let only the first two resources pass.

All output state connections from a gate are triggers; gates do not accumulate resources and therefore label modifiers, node modifiers and activators originating from a gate have no function. The triggers are activated instead of redirecting

⁶It can be convenient to think of a deterministic gate with conditional outputs as ‘counting gate’.

⁷This value can be set represent other types of random distribution if needed.

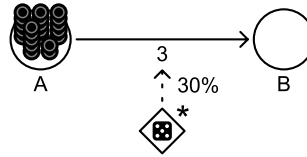


Figure 4.13: An automatic, random gate controlling the flow of resources between two pools. In this case there is a 30% chance three resource will flow from A to B every time step.

resources. These triggers can also be conditional or probabilistic. In this way gates can be used to control the flow of resources (see figure 4.13).

4.5 Four Economic Functions

In their discussion of a game's internal economy represented by the flow of resources, Adams and Rollings identify four basic economic functions: sources, drains, converters and traders (Adams & Rollings, 2007, 331-340). Sources create resources, drains destroy resources. Converters replace one type of resource for another, while traders allow the exchange of resources between players or game elements. In theory, a pool or combination of pools and gates can fulfill all these functions, but for clarity it is useful to introduce special nodes to represent sources, drains, converters and traders.

Sources are nodes that produce resources. In *RISK*, the building action is a source: it produces armies. Likewise passing 'Go' in *MONOPOLY* also is a source: it generates money. Health packs are sources of health in shooter games. As any node in a Machinations diagram, sources have an activation mode that is either passive, interactive or automatic. An example of an automatic source is the steady regeneration of the protective shields of the player's star fighter in *STAR WARS: X-WING ALLIANCE*. The rate at which a source produces resources is a fundamental property of a source. The building action in *RISK* and the passing of 'Go' in *MONOPOLY* are examples of sources activated by user actions and game events respectively. Adams and Rollings distinguish between 'limited' and 'unlimited' sources (Adams & Rollings, 2007, 333). A limited source can easily be represented as a pool without inputs that starts with a number of resources on it. An unlimited source can be represented as a pool without inputs but with a sufficiently large number of resources on it (or rather an infinite number of resources). To represent unlimited sources, the Machinations framework includes a special source node represented as a triangle pointing upwards (see figure 4.14). Note that Adam and Rollings' notion of a 'limited source' is still represented as a pool with no inputs.

Drains are nodes that consume resources. In an adventure game where you can cross hot lava at the cost of loss of health points, the lava acts as a drain. Being underwater in most games causes a resource representing breath to be drained away. The rate of a drain is determined by the flow rate of its input resource connection. Some drains consume resources at a steady rate while oth-

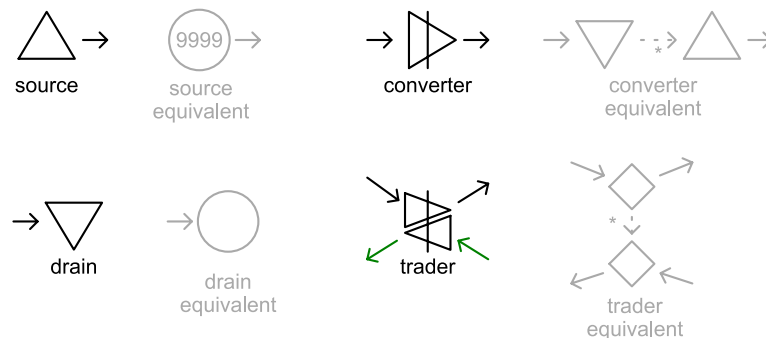


Figure 4.14: The four economic functions in a Machination diagram, with their equivalent constructions.

ers consume resources at random rates or intervals. Drains could be represented as a pool with no outputs, but the Machinations framework includes a special drain node represented as a triangle pointing downwards (see figure 4.14).

Converters convert one resource into another. In MONOPOLY the option to buy property acts as a converter: the resource money is converted into another resource: property. In a shooter game, killing enemies might also invoke a converter. In this case ammunition is used in an attempt to kill, which in turn, when the enemy is put down, might be converted in new ammunition or health packs dropped by the enemy. Converters act exactly as a drain that triggers a source, consuming one resource to produce another. As with sources and drains, converters can have different types of rates to consume and produce resources. The Machinations framework represents a converter as a vertical line over a triangle pointing to the right.

Traders are nodes that cause resources to change ownership when fired: two players could use a trader to exchange resources. The board game SETTLERS OF CATAN is built around a trading mechanism allowing players to trade the five types of resource cards among each other against exchange rates they establish among themselves. A player that has many of timber cards, might for example decide to exchange three timber cards for two wool cards of another player. Compared to converters, traders are relatively rare; in many games, what appears to be a trader is often implemented as a converter. For example, depending on the implementation, the merchants in many computer role-playing games where players can barter for goods and equipment are converters, not traders. These merchants will happily buy whatever the player has to offer and seem to have an unlimited supply of handy items and money to buy the player's unwanted loot (Castronova, 2005, 198-199). FALLOUT 3, where all traders' supplies are limited, is an exception. A trading mechanism can be constructed by two gates connected by a trigger ensuring that when one resource is received the other is returned in exchange. The Machinations framework represents a trader as a vertical line over two triangles pointing left and right.

The difference between converters and traders is not always immediately

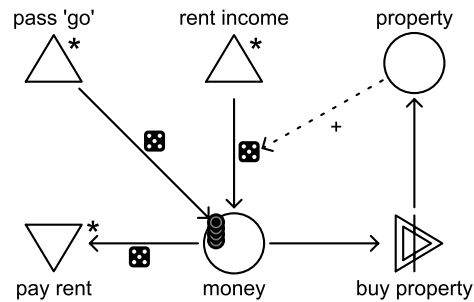


Figure 4.15: A Machinations diagram for MONOPOLY.

clear. Especially from the perspective of an individual player, converters and traders will almost have the same function: pass a number of resources to it, and get a number of other resources in return. Yet, there is an important difference. As pointed out above: a converter can be seen as a combination of a drain and a source. Using a converter resources are actually consumed and produced, and therefore the total number of resources in the game might change. Whereas with a trader the number of resources always stays the same (also see Adams & Rollings, 2007, 334).⁸

The economic function of a particular game element can change according to the perspective of the diagram. For example, figure 4.15 represents MONOPOLY from the perspective of an individual player. Only in the perspective of an individual player rent income is a source. From the perspective of the entire game rent acts as trader: money simply exchanges ownership. In this case, the bank should be modeled as an individual pool with which money is only traded. After all, the boxed game comes with a finite supply of play money. All of these would constitute valid models of the same game. As mentioned in the introduction, the Machinations framework allows one to model a game with different levels of detail and abstraction. The important question is: what does one try to model? For a basic understanding of behavior of MONOPOLY a simple, limited perspective as in figure 4.15 might suffice. Especially when one can imagine how this same structure is repeated for every player. But for a more thorough analysis more detail might be required.⁹

⁸When a converter is replaced by two pools as the equivalents of a source and a drain suggest the number of resources also stays the same. However, if one considers resources that are trapped on a pool without outputs and which state no longer affect the game to be *inactive* then the number *active* resources might change.

⁹For an example of such an analysis see http://www.jorisdormans.nl/machinations/wiki/index.php?title=Tutorial_1. In this example the model for MONOPOLY includes two players. The only reason I can imagine why anyone would want to model the bank is in a detailed study to find out how much money should be included in the published game.

4.6 Feedback Structures in Games

The structure of a game's internal economy plays an important role in a game's dynamic behavior and gameplay. Just as feedback plays an important role in any dynamic system (see section 1.5), it plays an equally important role in a game's internal economy. The idea of applying the concept of feedback to games is not new. During his 1999 lecture at The Game Developers Conference Marc LeBlanc introduced feedback loops to the game design world (1999). Since then, feedback loops have been discussed by a number of influential designers, including Salen & Zimmerman (2004), Adams & Rollings (2007) and Fullerton (2008). A classic example of feedback in games can be found in MONOPOLY where the money spent to buy property is returned with a profit because more property will generate more income. This feedback loop can be easily read from the Machinations diagram of MONOPOLY (figure 4.15): it is formed by the closed circuit of resource and state connections between the money and property pools. More specifically, for feedback to exist, a close circuit of connections is required that consists of at least one state connection that is not an activator. A closed circuit of resource connections can only create a loop of resources. To change the rate of the flow, at least one label modifier, trigger or activator that changes or triggers the production or consumption of the resources must be part of the loop. Note that for this reason a closed circuit of resource connections that includes a converter or trader also constitutes a feedback loop, as their equivalent constructions do include a state connection (see figure 4.14).

As is the case in classic control theory (DiStefano III et al., 1967), Marc LeBlanc distinguishes between two types of feedback: positive and negative feedback (also see section 1.5). Positive feedback strengthens itself and destabilizes a system. Positive feedback occurs when an effect fuels itself. In the MONOPOLY example above, the feedback is positive because investing money will generate more money. Positive feedback amplifies small differences between players: a player that has a lucky break early in the game will find this luck amplified over time: a player that by chance gets the option to get more money or good property early in a game of MONOPOLY is very likely to win.¹⁰ Positive feedback can be applied to 'positive' game effects but also to 'negative' game effects, as is the case with loosing pieces in CHESS, which increases the chances of loosing more pieces, and which will eventually make you lose. LeBlanc suggests that positive feedback drives the game to a conclusion and magnifies early successes (see also Salen & Zimmerman, 2004, 224–225).

Negative feedback is the opposite of positive feedback. It stabilizes a game by diminishing differences between players, by applying a penalty to the player who has done something that takes him closer to his goal and winning the game, or by giving advantages to the trailing players. Many racing games use negative feedback to keep a race close, either by giving trailing players more advantages or by hindering leading players. This effect is often described as 'rubber-banding'.

¹⁰Which, incidentally, is exactly the point the original designers of MONOPOLY's ancestor THE LANDLORD GAME were trying to make: it was a critique of capitalism that favors those that possess capital and works to widen the gap between the rich and the poor (Fron et al., 2007).

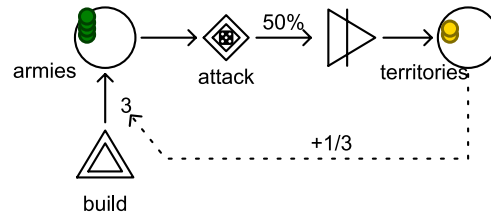


Figure 4.16: The core feedback loop involving armies and territories in RISK.

It can be implemented by blatantly giving trailing players better acceleration and more grip, or more subtly as is the case in *SUPER MARIO KART* by having the most effective weapons in the game affect cars in front of the player that uses them. LeBlanc points out that in most multiplayer games that allow direct interaction, some sort of negative feedback is already in place: rational players will target the leader more than any other player. As one might expect, negative feedback can prolong a game and magnifies late successes.

Control theory, in almost all cases, strives for negative feedback while avoiding positive feedback, as it aims to create stable systems. A large part of control theory concerns itself with determining and optimizing the stability of the system. For games the situation is, of course, very different. Positive feedback loops are much more frequent in games because, in general, players do not want to play a game that is stable and drags on forever. Negative feedback does have an application within games: most games with only positive feedback will seem too random to many players as they will be unable to catch the player who took an early lead; negative feedback is often used to balance out early successes.

Just as one feedback loop will only create weak emergence in Jochen Fromm's typology (see section 1.5), most games need multiple feedback loops to display truly interesting emergent behavior. The game *RISK* is an excellent illustration of this as in this game four feedback loops interact.

The core feedback loop in *RISK* involves the resources armies and territories. Figure 4.16 depicts this core feedback loop. The label '+1/3' of the label modifier that sets the output flow rate of the source 'build' indicates that the output of the source is improved by one for every three territories the player has.

The second feedback loop in *RISK* is formed by the 'cards' that are gained from a successful attack (see figure 4.17). Only one card can be gained every turn, thus the flow of cards passes through a limiter gate first. Collecting a set of three cards can be used to generate new armies. Not every set generates armies, and some sets generate more than others. In figure 4.17 these effects are indicated by the random symbol labeling the output of the converter that converts cards into armies.

The third feedback loop is activated when a player manages to capture a continent, which will give the player bonus armies every turn (see figure 4.18). In *RISK* predefined groups of territories form continents as indicated by the design of the game board. In the diagram this level of detail is not possible, instead

the construction is represented as a pool connected to another pool with a node modifier. In this particular case, seven territories will count as one continent which will in turn activate the bonus *source*.

Finally, the fourth feedback loop is activated by the loss of territories due to the actions of other players (figure 4.18). Which player is going to attack which other player is dependent on many factors, including those players strategies and preferences. Sometimes it is opportune to prey on weaker players in order to gain territories or cards, sometimes it is important to oppose stronger players to keep them from winning. In the diagram this is indicated by the multiplayer dynamic label (an icon depicting two pawns) affecting the resource flow to the drain on the right. The number of continents a player captured has a strong influence on this. As in general, players do recognize that other players that have a continent have a big advantage and will usually act against that more vigor. The player might also lose armies to actions of other players. I have chosen to omit them from the diagram to avoid too much clutter. It should not affect the argument too much. The important thing is, that in RISK there is some form of friction caused by other players, and the strength of this friction increases when the player has captured continents. This type of friction is a good example of the negative feedback that can almost always be found in multiplayer games where players can act against each other as pointed out by LeBlanc (see above).

The first three feedback loops in RISK all are positive: more territories or cards will lead to more armies which will lead to more territories and cards. Yet they are not the same. The feedback of cards is much slower that the feedback of territories as a player can get only one card each turn, but at the same time the feedback of the cards is also much stronger. Feedback from capturing continents operates faster and even stronger. These properties are important characteristics of the feedback loops that have a big impact on the dynamics of the game. Players are more willing to risk an attack when it is likely that the next card they will get completes a valuable set: it does not improve their chances of winning a battle but it will increase the reward if they do. Likewise the chance of capturing a continent can inspire a player to take more risk than the

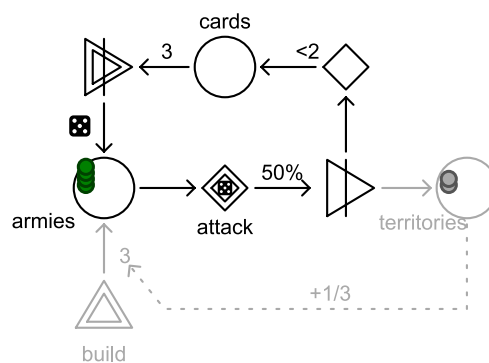


Figure 4.17: The second feedback in RISK involving cards and armies.

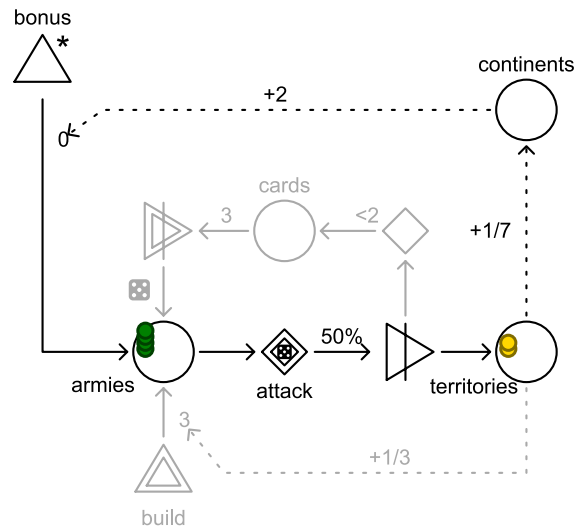


Figure 4.18: The third feedback loop in RISK: bonus armies through the possession of continents.

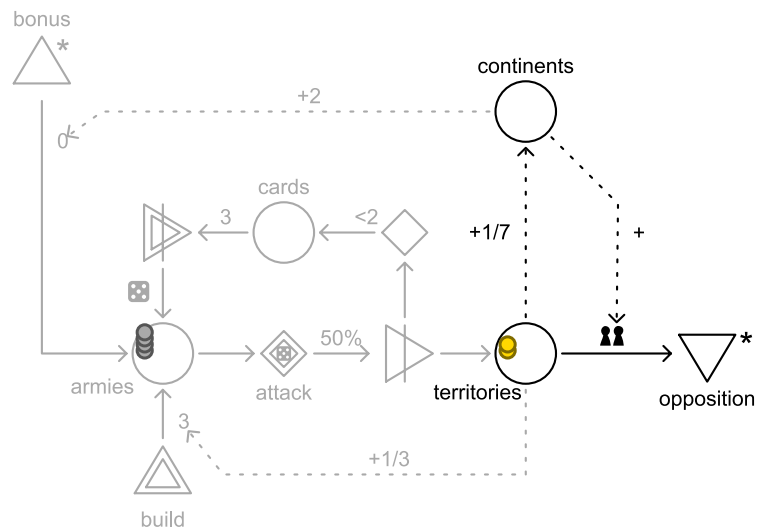


Figure 4.19: The fourth feedback loop in RISK: negative feedback caused by capturing continents. Note that in this figure the label modifier affecting the resource connection from the territories pool to the opposition drain does create a closed circuit as both nodes at the end and the start might be affected by the changing flow rate.

player should. In RISK the player's risks and rewards constantly shift, making the ability to understand these dynamics and to read the game a decisive skill in this game. These three positive feedback loops play an important role but simply classifying them as positive does not do justice to the subtlety of the mechanics.

4.7 Feedback Profiles

To really appreciate the feedback structure of RISK the differences between the three positive feedback loops must be explored in more detail. The feedback of capturing territories to be able to build more armies is straightforward, fairly slow and involves a considerable investment of armies. Often players lose more armies in an attempt to conquer territories than they will regain with one build. This leads to the common strategy to build during multiple, consecutive turns.¹¹ The feedback involving the cards requires a considerably larger effort on the part of the player. Players can only gain one card during a single turn, no matter how many attacks were successful. However, depending on the cards the player draws, the return might be much higher. The feedback involving continents is very fast and with a high return. Players will receive bonus armies every turn no matter whether they choose to build or to attack. The feedback is so strong and obvious that it will typically inspire fierce counter measures from other players.

Table 4.1 lists seven characteristics that, together with the determinability characteristic discussed below, form a more detailed profile of a feedback loop. At a first glance some of these characteristics might seem overlapping, but they are not. It is easy to confuse positive feedback with constructive feedback and negative feedback with destructive feedback. However, positive destructive feedback does exist. For example, loosing pieces in a game of CHESS will increase the chance of loosing more pieces and loosing the game. Likewise, the board game POWER GRID employs a mechanism in which the game leaders have to invest more resources to build up and fuel their network of power plants: negative feedback on a constructive effect.

The strength of a feedback loop is an informal indication of its impact on the game. Strength cannot be attributed to a single characteristic: it is the result of several. For example, permanent feedback with a little return can have a strong effect on the game. The effects of a feedback loops on a game can drastically change with these characteristics. Feedback that is indirect, slow but with a lot of return and not durable has a strong destabilizing effect. In this way even negative feedback can be used to destabilize a system if it is applied erratically or when its effects are strong, but slow and indirect.

In many games the profile of a feedback loop is also affected by factors such as chance, player skill and social interaction. Table 4.2 lists the different types on determinability used in the Machinations framework and the icons used to

¹¹In RISK players can choose to build or attack during a given turn; when they choose to build they will gain a fixed number of armies, when choosing to attack they can attack as many times as they choose during a turn. In RISK there is a rule that forces a player to attack after three consecutive turns of building.

Characteristic	Value	Description
Type	Positive	Amplifies differences, destabilizes a game.
	Negative	Dampens differences, stabilizes or balances a game.
Effect	Constructive	Operates on a game effect that helps a player win.
	Destructive	Operates on a game effect that will make a player lose.
Investment	High	Many resources must be invested to activate the feedback.
	Low	Few resources must be invested to activate the feedback.
Return	High	The net gain is high
	Low	The net gain is low
	Insufficient	The gain does not outweigh the investment (net gain is negative).
Speed	Immediate	The feedback is in effect immediately.
	Fast	The feedback takes a little time to take effect.
	Slow	The feedback takes a lot of time to take effect.
Range	Short	The feedback operates directly over a few steps.
	Long	The feedback operates indirectly over many steps.
Durability	None	The feedback works only once.
	Limited	The feedback works only over a short period of time.
	Extended	The feedback works over a long period of time.
	Permanent	The effect of the feedback is permanent.

Table 4.1: Feedback Characteristics





Type	Icon	Description
Deterministic	<i>(none)</i>	Given a certain game state, the mechanism will always act the same.
Random		The mechanism depends on random factors. The randomness can affect speed and/or return of a feedback loop, or the possibility of feedback occurring at all. It can create an infrequent return. Random feedback is difficult for the player to assess, and increases the chance of deadlocks.
Multiplayer-dynamic		The type, strength, and/or game effect of the mechanism are affected by the direct interaction between players.
Meta-dynamic		The type, strength, and/or game effect of the mechanism are affected by the tactical or strategic interaction between players.
Player skill		The type, strength, and/or game effect of the mechanism are affected by the player's manual skill in executing the action.

Table 4.2: Determinability

denote them. These icons can be used to annotate resource connections and gates in a diagram. A single feedback loop can be affected by multiple and different types of nondeterministic resource connections or gates. For example, the feedback through cards in RISK is affected by a random gate and a random flow, increasing its unpredictability.

The profile of multiplayer feedback in a game that allows direct player interaction, like RISK, can change over time. As LeBlanc already pointed out, it often is negative feedback as players act stronger, or even conspire against the leader. At the same time, it can also be positive as in certain circumstances, as mentioned above, it can be beneficial to prey on the weaker player.

The skill of players in performing a particular task can also be a decisive factor in the nature of feedback, as is the case for many computer games. For example, TETRIS gets more difficult as the blocks pile up, the rate at which players can get rid of the blocks is determined by their skill. Skillful players will be able to keep up with the game much longer than players with less skill. Here player skill is a factor on the operational or tactical level of the game. In games of chance, tactics, or games that involve only deterministic feedback, a whole set of strategic skills can be quite decisive for the outcome. However, that is a result of a player's understanding of the game's feedback structures as a whole, and as such it is not an element that can or needs to be modeled within the structure. This feedback loop in TETRIS is also affected by randomness. The shape of the block is randomly determined by the game. Although, the skill is

generally more decisive in TETRIS, the player just might get lucky.

Games that feature only deterministic feedback can still show surprising unpredictable outcomes, as emergence itself can be a source of unexpected and hard to predict behavior. In fact, it is my conviction that a well-designed game is built on only a handful feedback loops and relies on chance, multiplayer dynamic, and skill only when it needs to and refrains from using randomness as an easy source of uncertainty.

A feedback loop's characteristics and determinability form the feedback's profile. While a profile like this can be very helpful in identifying the nature of feedback in a game, it does little to reveal the interaction between different feedback loops. This is where diagrams, such as figure 4.20, excel. Many of the characteristics of feedback loops described above can be read from the diagrams. The effect of the feedback is directly related to the constructive or destructive nature of the feedback loop, whereas return and investment depends on the number of resources involved. A feedback loop that consists of almost only state connections and triggers, and few interactive nodes, is likely to have a high speed. Range can be read from the number of elements involved in the feedback loop, speed from the number of iterations required to activate the feedback. The return of a feedback loop must be read from the modifiers of the arrows that create the closed circuit, as some of these modifiers might be nondeterministic the return is more difficult to assess or actually becomes uncertain. The type of feedback (positive or negative) is perhaps the most difficult to read from a static representation, and requires careful inspection of the diagram, but this is possible, too. Note that the plus symbols in the diagrams do not indicate positive feedback, only that there is positive correlation between the number of resources in the pool and the label it is affecting. A positive correlation can induce negative or positive feedback.

4.8 Feedback Analysis and Recurrent Patterns

An analysis of a game's feedback loops can be used to identify structural strengths and flaws in its design. To create interesting and varied gameplay feedback is an important device, and most successful games incorporate two or more, but not that many more, feedback loops in its main structure. Structural flaws, or 'bad smells' in analogy to software engineering, are constructions that are best avoided. If we take RISK again as our example, we can identify one of its problems from play experience: building as often as you can is an effective, almost dominant, strategy. To counter this strategy the game includes a special rule preventing the players from building on more than three subsequent turns. Inspection of the feedback structure of the game suggests other ways of resolving the problem. Attacking feeds into a triple positive feedback structure (through territories, cards and continents), which is a strength of its design, but apparently the feedback is not effective enough. Strengthening the feedback of territories will help only a little as building is part of the same feedback loop and will probably encourage the unwanted behavior. Either the feedback through cards or the feedback through continents needs to be improved. The

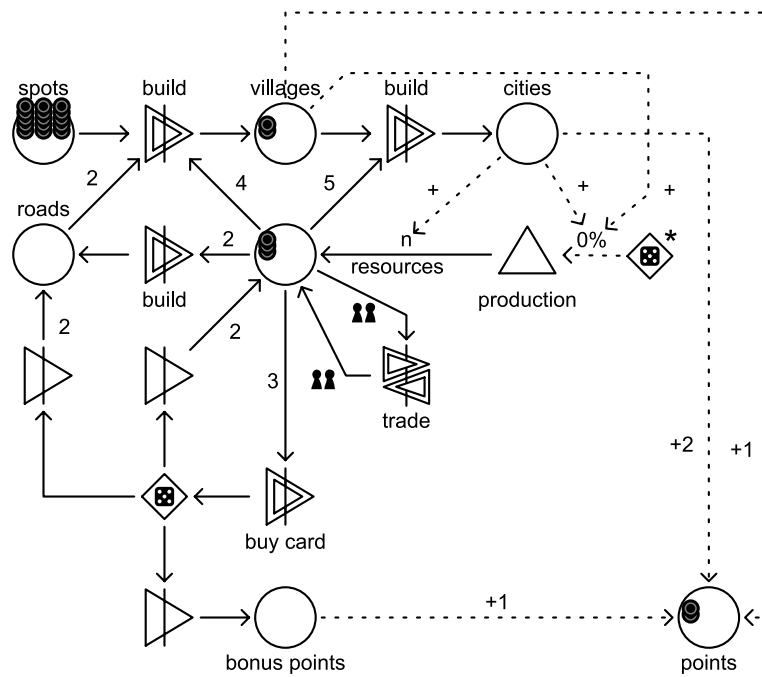


Figure 4.20: A Machinations diagram for *SETTLERS OF CATAN*, which is won by collecting ten points. The game’s five resources are collapsed into one for this diagram. Normally, a player has to pay with a particular set of resources to perform an action. The relative value of these different resources varies as production of each individual resource is subjected to chance. *SETTLERS OF CATAN* has three main feedback loops: 1) The slow, expensive but durable increase of production through the investment in roads, villages and cities. 2) Buying cards, which is fast, unpredictable and has no durability. And 3) through trade with other players which is subject to multiplayer-dynamics.

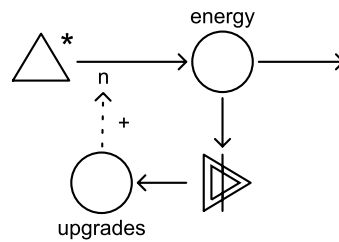


Figure 4.21: The dynamic engine pattern.

card feedback loop involves two random factors: success of attack and the blind draw of the card itself. This makes the feedback unpredictable and very hard for the player to assess. In general, involving too much randomness in the same loop is best avoided, especially when this randomness affects different steps in the loop. It is very hard to balance and predict the feedback of such a loop, so reducing the randomness, for example by allowing the winner a pick of three open cards, will help a lot.

Alternatively the feedback through the capture of continents can be improved. The problem with this feedback is that it has a high return, is permanent, direct and fast: it is very obvious and will inspire strong reaction by opposing players, in other words it acts as a red flag. Combined with a relative high investment, it constitutes an effective but risky strategy. The strength and the obviousness of the feedback invites a strong negative feedback. This creates a feedback loop that is too crude: it is either on and going strong or it is off. Either the player succeeds in taking and keeping a continent and has a very good shot at winning, or the player is hit hard and loses what usually is a considerable investment. By making the feedback less strong, and perhaps increase the number of continents (or rather regions) for players to conquer, a more subtle feedback loop is created that will pay-out more often without unbalancing the game too much.¹²

Looking at feedback structures in games, many recurrent patterns emerge. For example, both MONOPOLY and RISK share a similar structured, positive feedback loop that can be found in many other games as well. This pattern, which I call a dynamic engine, revolves around a source producing one type of resource, which can be converted to improve the source. Figure 4.21 depicts this elemental feedback pattern, using the generic names energy and upgrades for the two resources involved. In MONOPOLY these resources are money and property respectively, whereas in RISK these are armies and territories. The pattern can be found in many more games. In STARCRAFT the player invests minerals to build SUV units to mine more minerals.

SETTLERS OF CATAN (see figure 4.20) has a more complicated implementation of this pattern, one where a player needs to build roads before that player can build villages, and where villages can be upgraded to cities. In this case the

¹²On the other hand, the game is called 'risk' for a reason, risk taking is an intended part of the game play. How much risk is suitable for this game is also a matter of taste.

dynamic engine is also part of a engine building pattern (see appendix B).

A dynamic engine has a very typical gameplay signature. When play begins players will invest most of their energy in upgrades for a while. At a certain point, players start to use the energy elsewhere or, when that is the set goal for the game, simply collect it. When one plots the output of energy over time in a graph, this leads to a sharply cornered line (see figure 4.22). This signature is recurrent in all games that use a dynamic engine, although it might be obscured by the randomness or nondeterministic behavior caused by other feedback structures. For example, with one strategy in STARCRAFT called “turtling”, players invest a lot in building their base, before starting to build an offensive force to attack the enemy. When these players start attacking their offensive output is usually quite large. In the opposite strategy, called the “Zerg rush” after one of the game’s playable factions, the players invest little in their base, instead they start building offensive units as fast as they can in a bid to overpower their opponents before they have built up adequate defense (see figure 4.23). The effectiveness of the latter strategy depends on the speed of the attacking player, but also on the balance between offensive capabilities, defensive capabilities and the building costs of the units involved. Section 4.11 will discuss the balance between these two strategies in more detail.

Twelve more recurrent feedback patterns are described and discussed in appendix B.

4.9 Implementing Machinations Diagrams

The online Machinations tool does not only allow users to draw Machinations diagrams, it can also run diagrams.¹³ While running, the resources in a diagram flow from node to node and flow rates change according to their distribution. The digital version of the diagrams introduces an extra activation mode, two different modes the nodes can use to push or pull resources, three new types of nodes, and the concept of color-coded resources. All of these are discussed in this section.

The new activation mode digital Machinations diagrams introduce is the

¹³See the tool’s web page: <http://www.jorisdormans.nl/machinations>

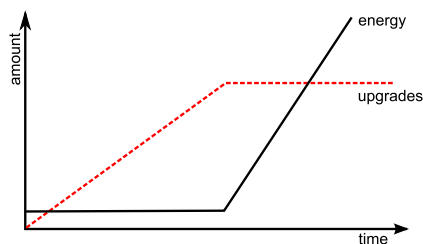


Figure 4.22: The gameplay signature of a dynamic engine.

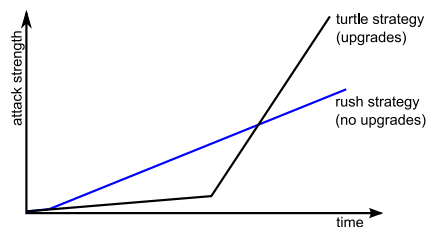


Figure 4.23: The turtle and rush strategies in STARCRAFT are the result of a dynamic engine.

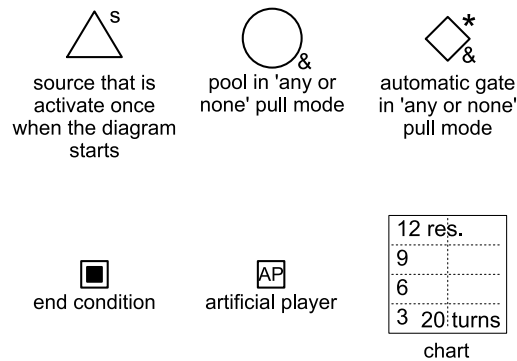


Figure 4.24: Extra modes and nodes in automated Machinations diagrams.

‘starting action’ mode. Nodes in this mode fire once when the diagram is started and are marked with an ‘s’ instead of the star used to mark automatic nodes (see figure 4.24).

The pull modes of a node specify in more detail how a node pulls resources from another node. There are two different pull modes:

1. By default, a node pulls as much resources as it can, up to the flow rates of its inputs. If not all resources are available, it still pulls those that are.
2. Alternatively, a node can be set to pull all or none resources. In this mode, when not all resources are available, none are pulled. Nodes that are in ‘all or none’ pull mode are marked with an ‘&’ sign (see figure 4.24).

These modes also apply to pushing modes: by default a pushing node sends as many resources out along its output resource connection up to outputs’ flow rate. A pushing node in ‘all or none’ mode only sends resources when it can supply all of its outputs. This means that nodes in push mode might be marked with both a ‘p’ and a ‘&’.

The three new nodes are: end conditions, charts, and artificial players. End conditions specify when a diagram has reached an end state and can stop further execution. Usually such a state is reached when a specified number of resources is collected or when a particular resource is completely drained (see figure 4.25). End conditions need to be activated through an activator, they do not have an activation mode as other nodes do. End conditions can be used to set goals or build simple timers to limit the game’s length. Diagrams that have end conditions are suited to ‘quick run’: instead of displaying the dynamic behavior as it develops over time, it runs the game to its completion immediately. Diagrams can also be run several times in succession, in this case the tool will show which end condition was reached how many times.

Charts can be used to plot the state of pools into a graph. Pools and graphs are connected using node modifiers, but to avoid visual clutter the tool represents these state connections as two small arrows, one leading out of the pool and one leading into the graph. The color of these arrows corresponds with the color of

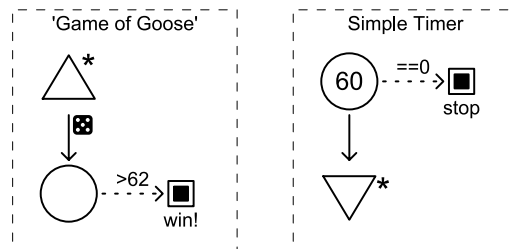


Figure 4.25: Two different end conditions.

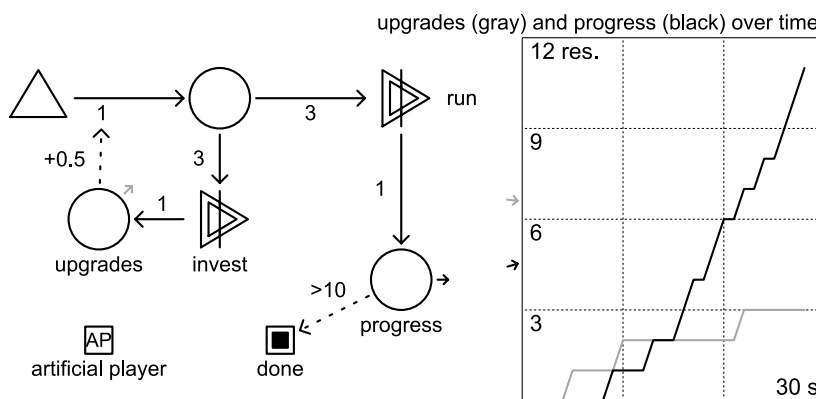


Figure 4.26: An example diagram that produces a chart and includes an artificial player.

the lines in the graph (see figure 4.26). The data collected by these graphs can be exported as simple comma separated values to be analyzed further by other tools.

Artificial players allow the use of the Machinations tool to simulate players interacting with the diagram. This introduces the possibility of automated multiple tests runs. The implementation of artificial players is rudimentary, but effective. Basically the artificial player has a list of options to activate a specified node and either goes through these options in sequence, or works down the list testing a specified probability for each option until it finds one node to activate. These options might be affected by the state of a pool. For example, the artificial player script for figure 4.26 reads:

```
invest = 100 - upgrades * 30
run = 100
```

This script will initially cause the artificial player to invest, but with every upgrade the chances it will invest are decreased by thirty percent. If it does not invest, there is a one-hundred percent chance it will run instead.

In the digital Machinations tool the color of resources is meaningful. If a resource connection has a different color than the color of the pool then only those

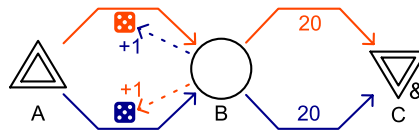


Figure 4.27: Automated Machinations diagrams allow the use of color-coded resources.

resources which color matches the color of the resource connection can be pulled through that resource connections. This allows different types of resources to be stored on the same pool. Likewise sources and converters producing resources, produce resources in the respective colors of their outputs, when these outputs have a different color than the source or converter. I use the term color-coding to refer to this use of colored resources and resource connections in a Machinations diagram. Label modifiers, node modifiers, and activators which color is different from the color of the node they originate from, act according to the number of resources of that color on the pool. Figure 4.27 illustrates how this can be used in a diagram. In this figure source A produces a random number of orange and blue resources every time it is activated. Both are collected at pool B, the number of orange resources on B increases the number of blue resources produced and vice versa. The user can only activate drain C when there are at least 20 red and 20 blue resources on pool C. Color-coded resources are used in the case-study of SIMWAR later in this chapter.

Digital Machinations diagrams offer the opportunity to collect data on the behavior of a game system before the game is built. It allows designers to test typical playing strategies. The artificial players do not have very advanced artificial intelligence, but they can still easily be programmed to follow certain strategies, and will happily do so over thousands of runs. As will become clear in the discussion of SIMWAR in section 4.11, this can be a valuable tool in identifying dominant strategies and testing the balance in a game. Artificial players can be activated and deactivated individually, allowing the user to define different artificial players set up to represent and experiment with different strategies within a single diagram.

4.10 Randomness and Nondeterministic Behavior

In many games, complexity is not the only source of nondeterministic behavior. As was argued in chapter 2, dice (or other random generators) are a good way to create nondeterministic behavior for those mechanics that are not the core of the gameplay. In this way, dice can be used to simulate the outcome of battles in RISK or KRIEGSSPIEL on the one hand, or to simulate the conditions that affect the rate of production in a game like SETTLERS OF CATAN on the other. From the perspective of the Machinations framework, randomness is a good tool to inspire particular behavior from the players but it might also be used to obscure dominant gameplay signatures that originate from certain feedback structures, such as the dynamic engine pattern.

An account on how randomness can affect the behavior of the player is given by John Hopson (2001). He argues that consistent with the findings in behavioral psychology experiments, player behavior is affected by chance and the interval the player is awarded for actions. When a player has a chance to be awarded at regular intervals, the player's attention and activity will spike at those intervals, where as when those intervals have random lengths, the player will be active most of the time. After all, the next action might lead to a new reward. As such, it is vital that in a Machinations diagram you can set up random values as well as random intervals.

The effect of randomness on the dynamic engine signature is illustrated with the following experiment. A simple racing game for two players utilizes a simple dynamic engine. The goal of the game is to collect thirty 'distance resources' by running. But the player can also choose to invest three energy to produce an upgrade which will increase the rate of production of energy (see figure 4.28). The energy production starts at a rate of 0.1 which means 1 resource is produced every ten seconds. Every upgrade improves this rate by 0.1. The artificial players controlling the black and gray diagrams are set up slightly differently. The black player will first buy four upgrades before it starts running. The gray player will buy only two upgrades. Obviously, black's strategy is superior: black wins every time. The chart in figure 4.28 shows this trend: the black line reaches 30 before the gray line does.

When the energy sources are changed to have a probable output of 10 percent every second, and each upgrade will increase this probability by another 10 percent (see figure 4.29), the pattern is broken. Figures 4.30 and 4.31 show sample data generated by different runs. Gray now has a chance of about 23 percent to win.¹⁴ In effect, the randomness can counter the effect of the feedback loop as was also suggested by Ernest Adams and Andrew Rollings (2007, 387).

The Machinations tool allows the designer control over random values produced. As we have already seen, the percent sign ('%') is used to denote a probability. A source that has probable output is labeled '20%' will have a twenty percent chance to produce a resource every time step. The tool can also simulate dice rolls by using a similar notation for dice rolls and calculations that is commonly used in pen-and-paper role-playing games. In these games 'D6' stands for a random number produced by a roll of a single six-sided die, where as 'D6+3' adds three to the same dice roll, and '2D6' adds the results of two six-sided dice and thus will produce a number somewhere between two and twelve. Other types of dice can be used as well: '2D4+D8+D12' indicates the result of two four-sided dice added with the results of an eight- and twelve-sided die. Unlike pen-and-paper role-playing games, the Machinations tool is not restricted to dice that are commercially available. It can use five-, seven or thirty-five-sided dice.

Intervals are created by using a slash ('/') for inputs and outputs. For example, a source that produces 'D6/3' resources will produce between one and six resources every three seconds. Intervals can also be random: a drain that has an input with a modifier that states '3/D6' will drain 3 resources with an interval

¹⁴I ran the diagram 1000 times which resulted in 768 wins for black and 232 wins for gray.

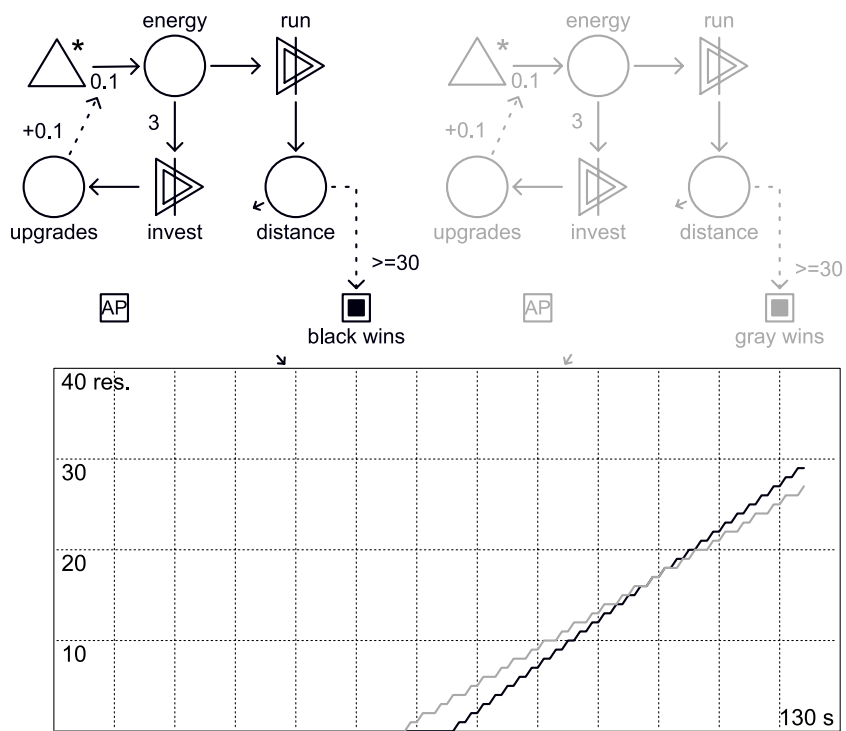


Figure 4.28: Two players racing for distance in a game with deterministic behavior.

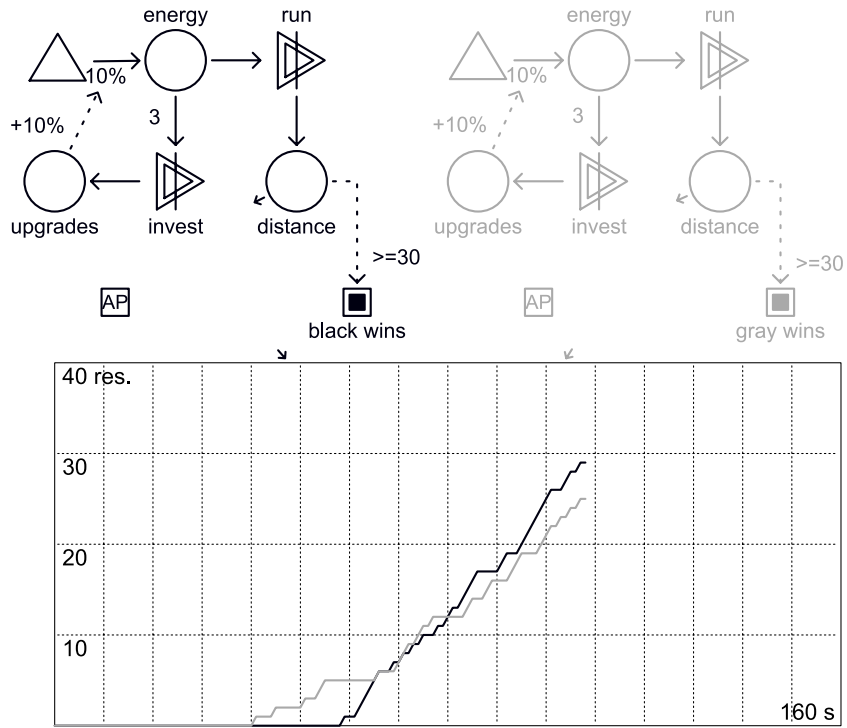


Figure 4.29: Two players racing for distance in a game with random behavior.

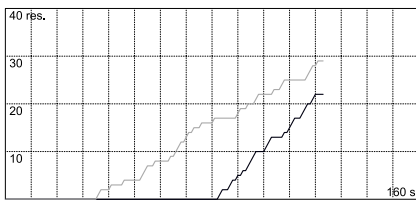


Figure 4.30: Sample result of a race with random behavior.

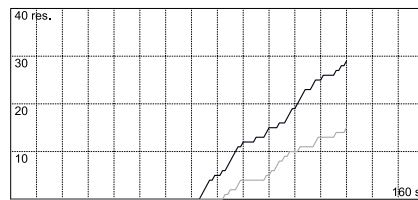


Figure 4.31: Another sample result of a race with random behavior.

between one and six seconds.

4.11 Case study: SIMWAR

The Machinations framework can be used to study existing games and support the design of new games. To illustrate the use of the framework I choose to discuss a game of some renown within the design community, yet has never been built. SIMWAR was presented during the Game Developers Conference in 2003 by game designer Will Wright, who is well-known for his published simulation games: SIMCITY, THE SIMS, etc. SIMWAR is a hypothetical, minimalistic war game that features only three units: factories, defensive units, and offensive units. These units can be built by spending an unspecified resource that is produced by factories. The more factories a player has the more resources come available to build new units. Only offensive units can move around the map. When an offensive unit meets an enemy defensive unit there is a fifty percent chance that one destroys the other and vice versa. Figure 4.32 can be seen as a visual summary of the game and includes the respective building costs of the three units. During his presentation Will Wright argued that this minimal real-time strategy game still presents the player with some interesting choices, and displays dynamic behavior that is not unlike the behavior found in other games within the same genre. Most notably Wrights argued that a ‘rock-paper-scissors’ mechanism affects the three units: building factories trumps building defenses, building defenses trumps building offensive units, whereas building offensive units trumps building factories. Wright describes a short-term versus long-term trade-off and a high-risk/high-reward strategy that recalls the ‘rush’ and ‘turtle’ strategies found in many real-time strategies (see section 4.8).

Building up a model SIMWAR using Machinations diagrams is best done in few steps. Starting with the production mechanism, a pool is used to represent a player’s resources. The pool is filled by an automatic source. The source’s production rate is initially zero, but is increased by 0.25 for every factory the player builds. Factories are built by clicking the interactive converter labeled ‘BuyF’, which will pull resources only when at least five are available. Figure 4.33 contains this diagram. The structure is a typical implementation of a dynamic

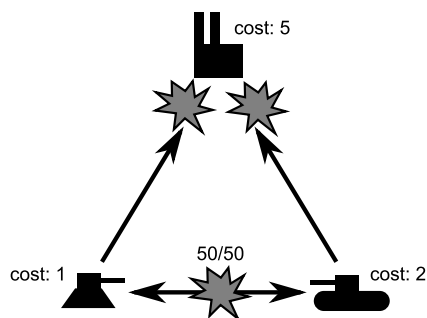


Figure 4.32: SIMWAR summary, after Wright (2003)

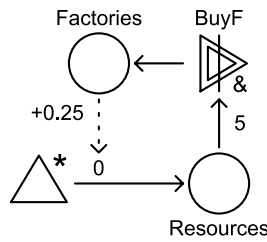


Figure 4.33: The production mechanism of SIMWAR.

engine pattern that we have seen before. As all dynamic engines, it creates a positive feedback loop: the more factories a player builds the quicker resources are produced which in turn can be use to build even more factories. Notice that, in this case, the structure requires players to start with at least 5 resources or 1 factory otherwise players can never start producing.

Resources are also used to buy offensive and defensive units. The mechanics for this are represented by figure 4.34. This diagram makes use of color coded resources. The resources produced by the converter labeled ‘BuyD’ are black while the resources produced by ‘BuyO’ are green as indicated by the color of their respective outputs. This means that black resources (representing defensive units) and green resources (representing offensive units) are both gathered on the ‘Defending’ pool. However, by clicking the ‘Attack’ gate, all green resources are pulled towards the ‘Attacking’ pool. Thus only offensive units can be used to launch an attack.

Figure 4.35 illustrates how combat between two players is modeled. Each attacking unit of one player (red on the left) increases the chance a defending unit of another player (blue on the right) is destroyed, and vice versa. In addition, attacking units increase the chance a factory is destroyed, but that drain is only active when the defending player has no defending units left.

Combining the structures of each step, a model can be created for a two player version of SIMWAR (see figure 4.36). One player controls the red and orange elements on the left side of the diagram, while the other player controls the blue and green elements on the right side of the diagram. Both sides are symmetrical. Note that, in contrast to figure 4.33, the supply of resources is

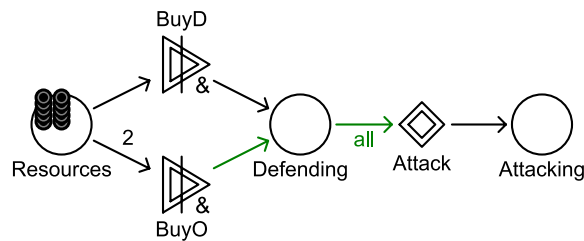


Figure 4.34: Offensive and defensive units in SIMWAR.

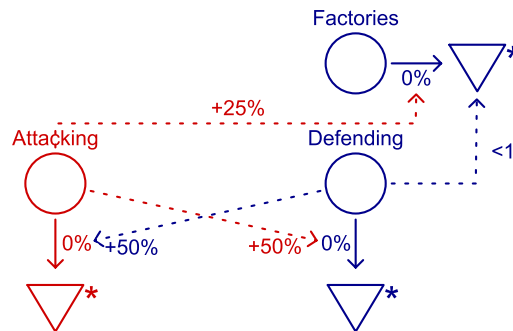


Figure 4.35: Combat in SIMWAR.

ultimately limited (as it is in most RTS games). This is to prevent the game from potentially dragging on for ever. If both players run out of resources before they managed to destroy the other, the game ends in a draw.

Figure 4.37 displays the relative strength of each player as it developed over time during a simulated session. The strength was measured by adding five for every factory the player owns plus one for each offensive and defensive units. The chart displays what might be called the fingerprint of an interesting match.

This particular session was played by two artificial players set up to follow what might be called a ‘turtling’ strategy, favoring factories and defensive units over offensive units. The script these players followed was:

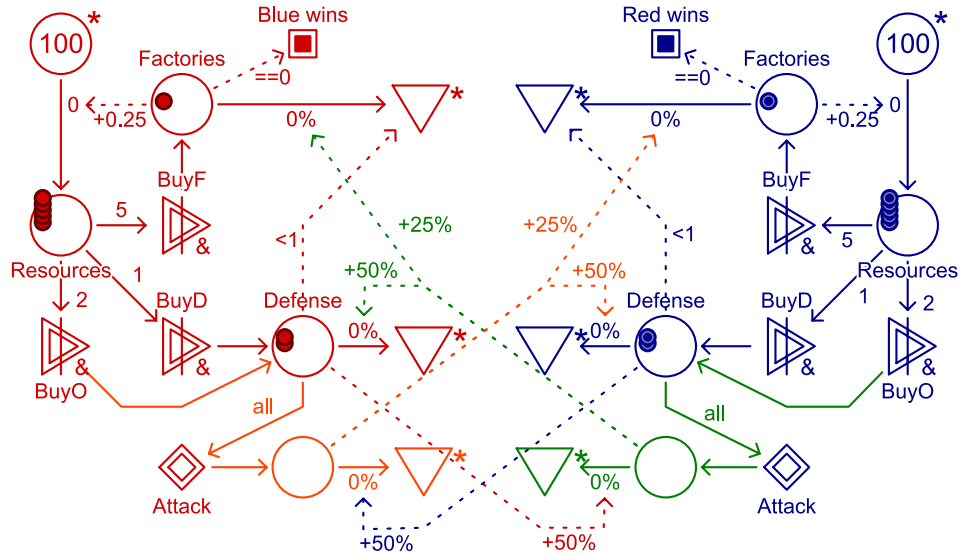


Figure 4.36: A Machinations diagram for SIMWAR. It features two players. One player controls the red and orange nodes on the left, the other the blue and green nodes on the right.

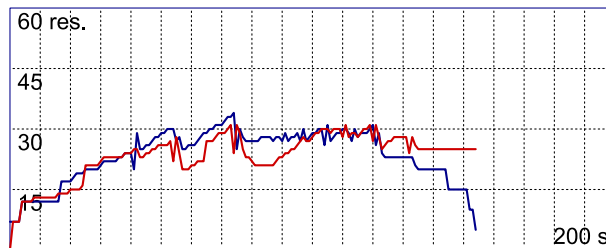


Figure 4.37: A chart showing the relative strength of each player as it developed over time during a simulated session eventually won by the red player.

```
Attack = Defense*5-30
BuyF = 100-Factories*30
BuyD = 100-Defense*15
BuyO = Factories*20+Resources*2
```

Another type of artificial player was created by setting up the script to follow a ‘rushing’ strategy, by building one factory before directing all resources towards building offensive units:

```
Attack = Defense*10-70
BuyF = 200-Factories*100
BuyD = 100-Defense*50
BuyO = 100
```

The ‘rushing’ strategy proved to be very unsuccessful. Figure 4.38 plots the strengths of both players over a typical session and also indicates when attacks were launched. The ‘rushing’ player (red) builds up a large attack, but does not recover once its units are lost. After that attack, it is fairly easy for the ‘turtling’ player (blue) to defeat red with a series of smaller attacks. Out of one thousand simulated session, the ‘rushing’ strategy managed to win only twice. Most published real-time-strategy games are balanced towards rushing strategies, as the latter tend to be harder to execute, and mastered later by players. In order to balance the game a number of ‘tweaks’ were tested: I increased the costs for factories and defensive units, and decreased the cost for offensive units and run the simulation one-thousand time for every modification (see table 4.3). Surprisingly, increasing the cost for defensive units seem to have little effect. Even when a defensive unit costs more than an offensive unit, making it a really poor choice, the turtle strategy remained dominant. This leads to the conclusion that the balance between rushing and turtling strategy is mostly affected by the balance between production and offensive units, and little by the balance between offensive and defensive units. Also note that increasing the factory costs initially increases the average game length, but decreases it for costs above eight. This can be explained by the fact that increasing factory cost slows the game as it takes more time to build up production capacity. At the same time a very high factory cost favors the rushing strategy, which tends to win faster than the turtling strategy. At higher factory costs the second effect dominates the first effect.

4.12 Conclusions

The Machinations framework formulates a clear theoretical vision on the structure of game mechanics and quality in games. Within the framework, gameplay is an emergent property of the system of rules. Machinations diagrams visualize those structures that directly contribute to emergent gameplay. Quality is attributed to the structure of the game mechanics, or rather to the feedback structures that are present within game mechanics. Up until now, feedback loops in games have been characterized as being either positive or negative.

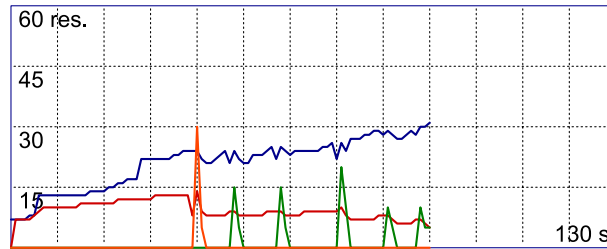


Figure 4.38: A chart showing a rushing player (red) against a turtling player (blue). The orange spikes indicate attacks waves launched by red, the green spikes indicate attack waves launched by blue.

Tweak	Avg. Time	Rush Wins	Turtle Wins	Draws
No tweaks	69.79s	2	997	1
Factory cost 6	76.33s	7	993	0
Factory cost 7	88.22s	107	889	4
Factory cost 8	91.86s	312	676	12
Factory cost 9	83.58s	557	426	17
Factory cost 10	68.03s	745	228	27
Offence cost 1.5	65.60s	164	836	0
Offence cost 1.4	54.72s	439	561	0
Offence cost 1.3	45.72s	583	417	0
Offence cost 1.2	32.73s	806	194	0
Defence cost 1.5	70.02s	16	983	1
Defence cost 2.0	74.35s	88	911	1
Defence cost 2.5	74.12s	196	803	1

Table 4.3: Tweaks to SIMWAR’s economy and the effects for ‘turtling’ versus ‘rushing’ strategies.

However, in order to explain gameplay we need more characteristics of feedback. The Machinations framework proposes to characterize feedback loops by their type, effect, speed, range, investment, return and durability. In addition, several types of nondeterministic flows can affect a feedback loop. The delicate interaction between multiple feedback loops must be taken into account to get a complete picture of the dynamics of games as complex machines that generate an internal economy. Patterns such as the ones discussed in this chapter codify structures that have proven to be successful in the past.

Although the Machinations framework utilizes a number of key concepts and terms, it is not a design vocabulary that needs to be expanded in order to include previously unencountered structures. Machinations diagrams work with only a handful of elements which can be combined in infinite constructions to capture just about any game. When these basic elements are understood, a designer should be able to read any Machinations diagram. The framework offers a high range of expressiveness for little investment on the part of the designer.

Machinations diagrams have an exact and consistent syntax. This means that the diagrams can be interpreted by a computer. In fact, the Machinations software tool implements diagrams. In other words, the diagrams can be run: they are interactive and dynamic, just like the games they are modeling. It allows models of games to be tested and explored quickly and efficiently. The tool even allows the designer to quickly gather quantitative data from simulated play sessions, offering a high and concrete return. Unfortunately this dynamic, interactive property of the software tool does not translate to paper; the interactive tool, and many of the examples discussed in this chapter, can be found on the Machinations web page: <http://www.jorisdormans.nl/machinations>.

The Machinations framework is a design tool first and foremost, but it can be used to record existing and non-existing games equally well. For pragmatic reasons many of the examples discussed in this chapter are models of well-known, existing games: it is hard to show the relation between rule structures and emergent gameplay when the reader is unfamiliar with the latter. In practice, using the Machinations tool allows a designer to simulate and run a design many times before building a prototype. It can also be used to track down flaws and suggest improvements for prototypes and published games. This applications of the Machinations framework should help the designer to get more out of each iteration in a play-centric design process.

The list of feedback patterns presented in appendix B and on the accompanying website is neither definitive nor complete. It is best to consider this framework as a set of building blocks that can be used to build an infinite number of different structures, some of which are recurrent patterns that can be used to analyze existing games and explore new concepts alike.

There are some limitations to the use of Machination diagrams. The idea of internal economy suits some games better than others. In particular, it works very well for board games, strategy games and management simulation games. Games that rely more on level design and mechanics of progression are addressed in the following chapter. In games where economy is more abstract it can be difficult to determine the best level of abstraction and scope for the model, as

is the case for games such as CHESS and GO. Those games seem to derive their emergent behavior less from an internal economy and more from the mechanics that govern tactical maneuvering, that fall outside the scope of this dissertation. Many games can be diagrammed in multiple ways depending on the designer's focus and the diagram's particular perspective. Still, feedback loops can go a long way in explaining the flow of a game, and should be consistent features even with different levels of abstraction and different perspectives.

Storytellers in all media and all cultures are, at least partially, in the business of creating worlds.

Scott McCloud (2000, 211)

5

Mission/Space

All games have rules and mechanics, but different types of games utilize different types of mechanics. As was discussed in Chapter 1 mechanics to govern physics, economy or levels are of a different nature and therefore must be treated differently. The previous chapter discussed the Machinations framework, diagrams and tool to represent discrete game mechanics that govern a game's internal economy, and to correlate their structure to emergent gameplay. It is a framework that works well for certain types of games. As was pointed out, the Machinations framework does not incorporate level design. Yet for level-driven games, such as most action-adventure games and most first-person shooter games, quality level design is critical. Unfortunately, level design has been studied less extensively than game mechanics dealing with physics or economy.

Level design is one of the aspects of game design where the different disciplines of art, design and technology converge. Level design combines all of these aspects as it works with core mechanics and art to create a spatial and temporal experience within the technological boundaries of the game's software-architecture. Level designers are usually responsible for turning gameplay elements and art assets into game spaces that create a concise and compelling experience. Often this requires scripting of simple behavior to create mechanics to control player progress. The specialized role of the level designer was one of the last roles to emerge from the growing game industry (Byrne, 2005; Fullerton, 2008). As a result there is no definite set of principles guiding level design yet, although a few common patterns and strategies can be found in different sources.

In this chapter I will present the Mission/Space framework. This framework provides a structural perspective on levels and players' progression through them. However, this chapter starts with investigating a few existing structural models for level design that are suggested by others in the game development community. The first section presents an overview and a discussion of common typologies for level layouts. The second section explores the models that focus on levels as a set or series of challenges to be overcome by players. The Mission/Space framework builds on these two different perspectives and includes separate graph representations for both the mission and space that comprise a level. With a detailed analysis of a game level from *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* I will illustrate and discuss the relation between the missions and

spaces. This discussion extends into the final sections of this chapter where I will show how the Mission/Space framework can be leveraged and expanded to get a better grip on mission, spaces and their relation. In chapters 6 and 7 I will combine the Mission/Space framework with the Machinations framework in order to integrate structures of emergence and progression more closely, and to unify both in an formal approach to game design that is based on model transformations, respectively.

5.1 Level Layouts

Level design has not been studied extensively. Yet, it is generally acknowledged that levels benefit from having a relatively simple gestalt or purpose, especially when this gestalt matches the intended gameplay, rhythm and pacing. To this end, a number of scholars of games and interactive storytelling categorized spatial structures frequently found in games. An overview and comparison of four of these typologies is presented in figure 5.1. This figure lists all the categories in each typology horizontally. Their relative position and size on the horizontal axis suggests overlap between the categories of each typology. Figure 5.2 illustrates each category with a schematic representation of their structure.

Of these typologies, Marie-Laure Ryan's specifically concerns itself with interactive story structure, while the others concern themselves with game levels. Still, the similarities between many of the structures they describe is striking. In fact, it is a common observation that in games, stories are, at least partially, structured spatially instead of temporarily (Jenkins, 2004). This causes some confusion whether these categories concern themselves with level geography or topology. As Ryan focuses on interactive storytelling, her categories are clearly topological, but the other three typologies are much more geographical in nature. From these typologies it appears that in level design topological structures and geographical structures are frequently isomorphic or at least often treated as such.

The different categories in the typologies in figure 5.1 can be grouped in to three super-categories. The categories on the left all concern tightly controlled structures that are fairly linear and directed; players usually cannot go back in these structures. Categories that describe railroading and branching techniques fall in this group. The categories in the center are all networks of some sort. These categories offer players more freedom to explore the world but still restrict players' positions to a restricted set. The categories on the right feature more open world that impose few restrictions on player movement.

One observation that can be made from figure 5.1 is that, although there definitely is some overlap between the different typologies, their differences are also quite prevalent. What is a main category in one typology might be a special case of a category in another. For example, 'bottlenecking' with Byrne can be regarded as a special instance of 'parallel structures' with Adams and Rollings. Categories present in most typologies are omitted in another: Adams and Rollings have no category that covers the 'tree/branching structures' with

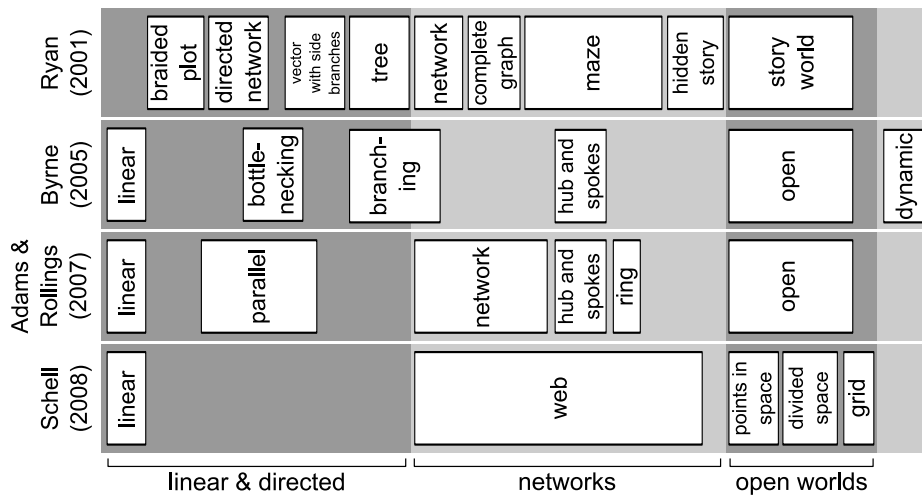


Figure 5.1: A comparison of four level typologies, based on (Ryan, 2001; Byrne, 2005; Adams & Rollings, 2007; Schell, 2008)

Ryan and Byrne. Other categories, such as ‘dynamic levels’ with Byrne and ‘grids’ with Schell, probably have no place in the typology at all, as they are separated by their mode of creation and not their topological structure.

The categories within some typologies do not appear to be on the same level as some other categories within the same typology. For example the ‘hub-and-spoke’ structure can be regarded as a special case of ‘network layout’ in Adam and Rollings typology. It might be a quite common structure in games, but it does not capture nearly as many different structures as the network layout does. At the same time, other specialized structures can be found that do not feature in any of these typologies. For example, a variation of the ‘hub-and-spokes’ layout: the ‘hub-with-loops’ layout, in which the player progresses through longer linear and directed level sections before returning to the hub can also frequently be encountered in games (see figure 5.3). Three of the nine main dungeons in *THE LEGEND OF ZELDA: TWILIGHT PRINCESS*¹ have this type of layout: Link, the game’s protagonist, can often quickly return to a central location after activating the trigger that unlocks a new section. The other six main dungeons have a normal hub-and-spoke layout. The point is, that from the typologies presented in figure 5.1, one cannot be certain to classify the hub-and-loops structure as a special combination of hub-and-spokes and linear structures or as a category of its own. The typologies do not provide enough guidance to safely argue one way or the other.

These problems indicate that understanding levels in games is not as straightforward as it may seem. The reason for this may be, as is one of the main points of this chapter, that levels do not constitute a single structure: they contain

¹This includes the ‘Hyrule Castle’ level that builds up to the final conflict with the game’s main antagonist.

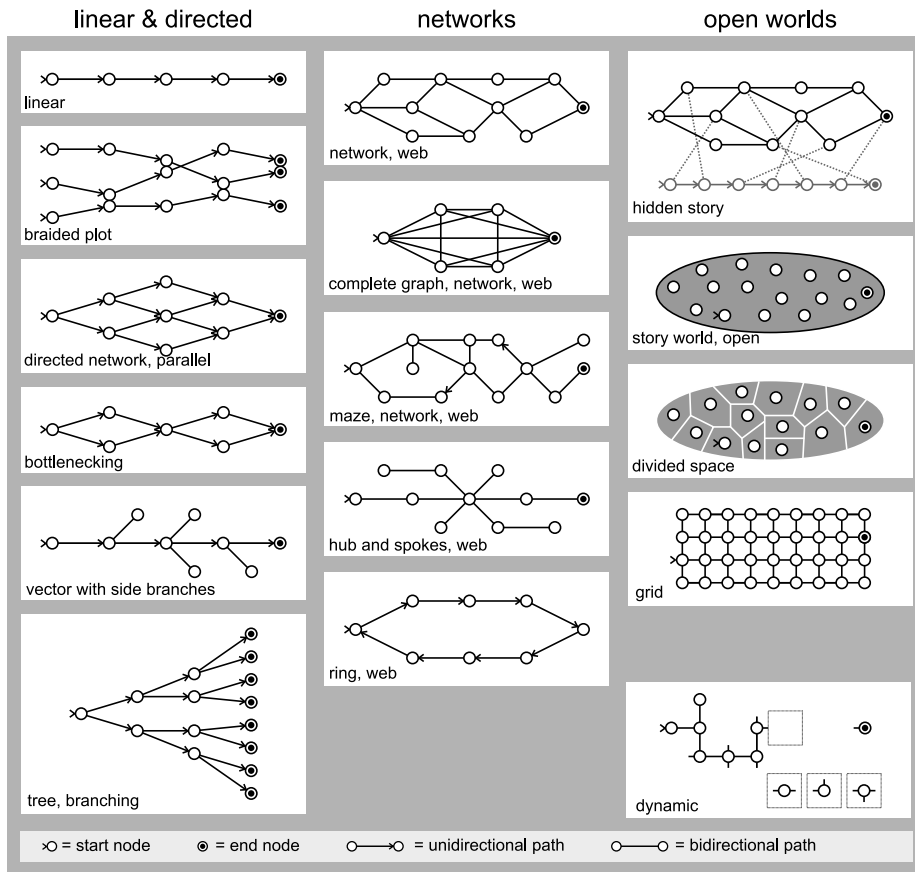


Figure 5.2: Illustrations of the level typologies found in (Ryan, 2001; Byrne, 2005; Adams & Rollings, 2007; Schell, 2008)

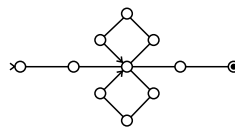


Figure 5.3: A hub with loops structure

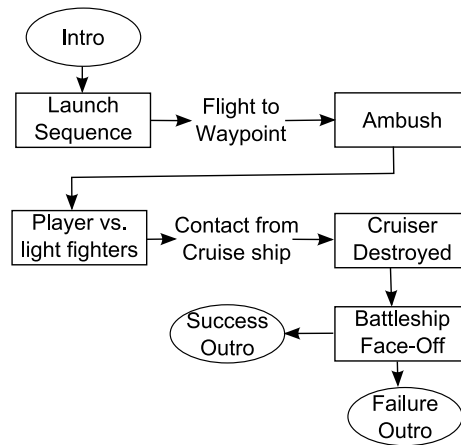


Figure 5.4: Cell diagram representing level structure, after Byrne (2005, 69).

at least a geography and a topological structure. Separating a level's topology from its geography will help to create much clearer perspective on game levels. In this respect, I consider the layouts discussed above to be geographies. A level's topology would focus more on the logical structure of the player's tasks.

5.2 Tasks and Challenges

A level's topology concerns itself with the tasks and challenges players must perform in order to complete a level. These tasks or challenges are usually fairly straightforward tests of the player's abilities. They might take the form of puzzles, fights, traps or hidden objects that need to be collected. Ed Byrne suggests structuring these tasks in cell-diagrams outlining the game's flow and highlighting the different player tasks (see figure 5.4). These cell diagrams are simple informal structures that read almost like a storybook that help design a level's layout or rhythm. Cell diagrams focus on a game's logical and temporal structure instead of its spatial layout (Byrne, 2005).

The temporal dimension of games and the players experience is also emphasized by the approach of Ben Cousins (2004). Cousins introduces the idea of a hierarchy to order the gameplay experience. In a detailed analysis of the game SUPER MARIO SUNSHINE he divides the experience in five layers, where the top layer constitutes the whole game. The subsequent layers describe the individual missions, mission elements, input elements, and primary elements. The middle layers intuitively correspond to player plans and intentions. In cognitive science they would have been called basic level categories for the games actions; these actions are most accessible to players and would typically be the actions players would use when describing the gameplay. The lowest layer represents the actions made possible by the game mechanics and interface; they correspond to the buttons pressed by a player and the resulting actions of the avatar, such as jumps or simple moves. Cousins stresses the importance of the quality of these

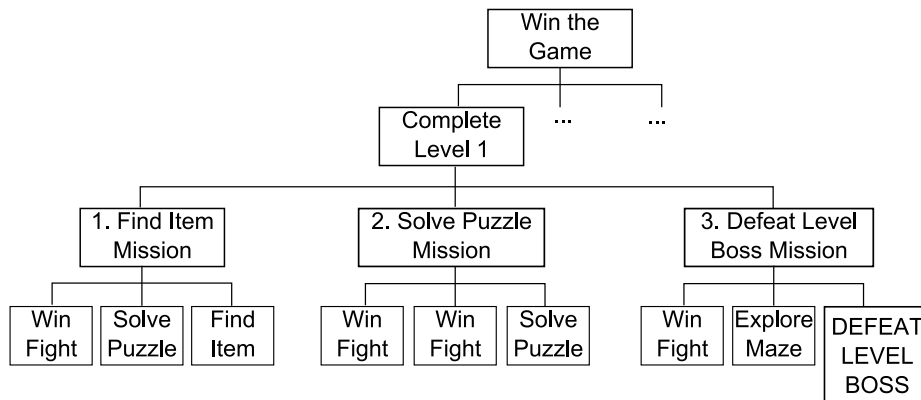


Figure 5.5: Cell diagram representing level structure after Adams & Rollings (2007, 282).

low level actions as the player spends much time performing them. Using simple metrics Cousins shows that 55 percent of the primary elements in a four minute session of SUPER MARIO SUNSHINE consisted of running forward and changing direction.

The ‘hierarchy of challenges’ described by Ernest Adams and Andrew Rollings was directly inspired by Cousins’ analysis. However, where Cousins focuses on individual sessions, with the hierarchy of challenges Adams and Rollings abstract away from individual sessions and represent all possible trajectories in a single image. In this hierarchy all the game’s challenges are ordered into a layered structure representing what a player needs to accomplish to complete the game (see figure 5.5). To build a hierarchy of challenges, inside knowledge of the level’s design is required. Some challenges can be performed in different order or even simultaneously. At the lowest level in the hierarchy are the atomic challenges: the micro actions the player needs to perform to get ahead. At higher levels in the hierarchy there are goals of individual sections and missions. At the highest level the game’s ultimate goal resides. Adams and Rollings discuss the different needs of visibility among the levels: games need to be very clear in their high level and atomic level challenges while they can be less clear for intermediate challenges. They also stress that presenting a player with simultaneous atomic challenges considerably increases the game’s difficulty (Adams & Rollings, 2007).

Creating simultaneous or alternative options in the hierarchy of challenges leads to a potentially more varied gameplay. However, this is more difficult in the higher levels of the hierarchy than it is in the lower levels (see also Arinbjarnar & Kudenko, 2009). DEUS EX is a good illustration of this. In this game the player has to get through a series of connected missions. At most low level challenges the player is presented with options: to deal with a guard the player might use stealth or violence. As players progress they choose how the game’s protagonist develops his skills: players can choose to specialize in different strategies to

overcome common challenges, such as the the use of force or stealth. Yet, at the high end of the challenge structure there are only few options and branches. It is only at the very end of the game that the player gets to choose between one of three alternative endings.

5.3 The Mission/Space Framework

The two different approaches, focusing on the geographic layout of a level on the one hand and on the sequence of tasks on the other, suggest that in a level both structures exist at the same time. These structures are superimposed onto each other and as a result it is all too easy to confuse one with the other or to take their interrelation for granted. The Mission/Space framework formalizes both perspectives and foregrounds that spaces and missions coexist within levels; a level has a particular geometric layout *and* a series of tasks that need to be performed in that space. At a first glance this observation seems obvious, and in fact, the level layouts and the hierarchy of challenges as discussed above seem to acknowledge a similar distinction. At the same time, many designers seem to forget about this distinction: for many games the mapping between the mission and the space is quite direct and their structures often are quite similar, even isomorphic. This section formalizes the structures of missions and spaces in games into a single framework. The advantages of this framework, where missions and spaces are treated as separate but related structures, is that their individual structures, but also their relations, can be investigated with much more clarity. The relation between mission and space is more sophisticated than a superficial survey would suggest. Games might reuse the same space for different missions, as is the case in *SYSTEM SHOCK 2* where the player traverses the same areas of a spaceship multiple times. *SYSTEM SHOCK 2* shows that the same space can accommodate multiple missions (assuming that the individual mission structures do not resemble each other too closely). Reuse of game space in this way is often economic: the developer does not have to create a new space for every mission in the game. It has gameplay benefits as well. For example, players can use previous knowledge of the space to their advantage, adding to the their sense of agency and the depth of the gameplay.²

A designer of a level works with both missions and spaces, just novelist work with plots (the sequence of events) and their narration (the way these events are told).³ During the design process, the designer is likely to loop between mission constraints and spatial affordances while setting up a delicate balance between

²Agency is commonly used to describe a player's power to affect and influence a game world. Agency is a core concept in Janet Murray's book *Hamlet on the Holodeck* where she defines it as follows: "the satisfying power to take meaningful action and see the results of our decisions and choices" (1997, 126).

³With plot and narration I refer to the similar notions of "story" and "narration" used by Gérard Genette (cf. 1980, 29). However to avoid confusion between the common sense use of the word "story" and Genette's more technical use I use replaced it with "plot". To add to the general confusion, Foster uses the terms "plot" and "story" to indicate a similar distinction, but here "story" means narration, and is completely opposite to Genette's use of the word (Foster, 1962).

the two that ultimately facilitates the target gameplay. For players, the space is the access point: in space, through play, they realize a mission.

Mikhail Bakhtin's notion of the 'chronotope' provides another interesting parallel for the combined mission and spaces in games. The chronotope refers to the artistic relation between time and space in literature. Bakhtin observes that in literature time and space are "fused into one carefully thought-out, concrete whole" (Bakhtin, 1981, 84), and that their particular artistic intersection is important to the literary form. He also observes that particular chronotopes correlate to particular literary genres. The first genre Bakhtin describes is that of Greek Romance. The plot typically involves two lovers who have to overcome many obstacles in order to marry. This requires many adventures that take place in several far-away countries. The intersection of plot-structure and the journey through hostile environments is surprisingly uniform across many stories from this genre. For games it is the artistic intersection between mission and space that has a similar, large impact on the experience. It also seems that particular configurations of missions and spaces correspond to particular game genres. For example, the mission/space configurations found in action-adventure games are usually far more linear and restricted than the configuration of an open-world with multiple quests that is typically found in computer role-playing games.

Often the complete, combined mission/space structure of a level is set up to generate a narrative experience. For example, the Forest Temple level in *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* is set up to generate play trajectories that resemble elements of the hero's journey as described by Vogler (2007). As Link enters the temple, his mission is set up as he finds the first monkey of eight monkeys he needs to liberate. Shortly after this he encounters a large spider guarding the first hub in the level. Defeating this spider grants access to many locations in the first part of the level; the spider acts as a threshold guardian and marks the transition into the dungeon realm of adventure. What follows are many tests and obstacles, during which our hero Link meets many new friends and enemies. Halfway through the level, Link's confrontation with the monkey king fulfills the same role as "approach to the inmost cave" where the hero confronts and fights the adversary or an important henchman for the first time. The hero escapes with the "elixir" that ultimately helps him to defeat the main adversary in a final climactic confrontation. In this case that elixir is the 'gale boomerang': a magic device that acts as a weapon and a key at the same time. With the gale boomerang Link can activate triggers that unlock the second part of the dungeon that hides the final level-boss. Just as the same structure never seems to grow stale for fairy tales and adventure films, it is a recipe that can be found in many of Link's adventures in this game or any game within the series. It is also found in many Mario games, Nintendo games, and many other games.

More importantly, the Forest Temple level illustrates how both mission and space are used to shape the play experience. The relation between the mission and the space is less straightforward than it may seem. In section 5.6 I will investigate the structure of the Forest Temple in more detail. First, I will discuss how the Mission/Space framework uses two kinds of graphs to represent missions and spaces respectively. These graphs have their own language and structures

that are discussed in the next two sections.

5.4 Mission Graphs

Mission graphs represent the players' progress towards a goal not by tracking their physical location, but by tracking the tasks they must perform to finish a level. A mission graph is a directed graph that represents a sort of to-do list with each node representing a task that might or must be executed by the players. Nodes can be in one of three different states; the current state of a mission is determined by the respective states of its tasks:

1. A task can be *available*. When a task is available the player can execute it.
2. A task can be *unavailable*. When a task is unavailable the player cannot execute it.
3. A task is *completed* when the player has successfully executed the task.

Edges in the mission graph determine how changes in tasks' states affect the state of other, subsequent tasks. There are three types of edges between tasks (also see figure 5.6):

1. A *strong requirement* indicates that the completion of the source task (called a strong prerequisite) is a necessary condition for the availability of the target task. A strong requirement is represented as a double arrow directed towards the target task.
2. A *weak requirement* indicates that the state of a target task is changed from unavailable to available when the source task (called a weak prerequisite) is completed *or* available, unless this is barred by the incompleted state of a strong prerequisite of the target task, or when this is barred by explicit inhibition (see below). A weak requirement is represented as a single arrow directed towards the target task.
3. *Inhibition* indicates that the state of a target task is set to unavailable when at least one the source task (called an inhibitor) is completed, unless the state of the target task is completed. Inhibition is represented as a solid line that ends in an open circle at the side of the target task.

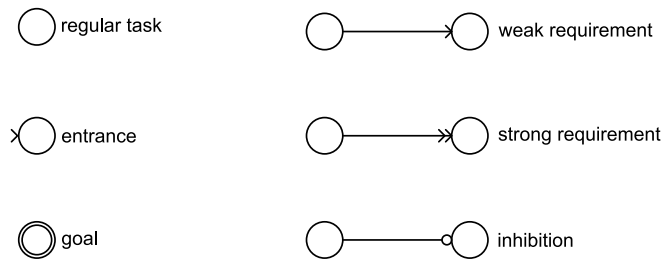


Figure 5.6: Basic elements of mission graphs

When a task has multiple predecessors, its state is determined as follows:

If it is completed, then its state no longer changes.⁴

Else, if it has inhibitors, then its state is set to unavailable when *at least one* of the inhibitors is completed.

Else, if it has strong prerequisites, then its state is set to available when *all* strong prerequisites are completed.

Else, if it has weak prerequisites, then its state is set to available when *at least one* of the weak prerequisites is completed.

Else, its state is set to unavailable.

There are three classes of tasks in a mission graph (also see figure 5.6):

1. *Regular tasks* are represented as circles. Different colors and letters can be used to represent specific (types of) tasks.
2. An *entrance* is a task that serves as an entry point for a player. A mission graph should have at least one entrance, but might have several entrances representing different starting points for the mission. Entrances cannot have prerequisites and one entrance starts in the available state to represent the player's point of entry.⁵ Entrances are represented as circles that are marked with an arrow head pointing towards it.
3. A *goal* is a task that finishes the mission when it is completed. A goal cannot be a prerequisite or an inhibitor for another tasks. A mission can have multiple goals in which case it is finished when one of these goals is completed. Goals are represented as circles that have a double outline.

⁴Curiously, in games it is fairly uncommon to be able to undo a task, which might make sense when that task inhibits other tasks. Allowing a special undo edge that could revert the completed state of a task would allow mission graphs to incorporate the option of allowing players to overcome problems they created for themselves. This notion, tantalizing as it may be, is left unexplored in this dissertation.

⁵In the case of multiple entrances, what entrance is activated depends on the player's or the game's actions prior to that mission.

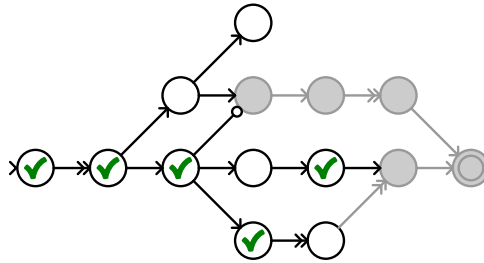


Figure 5.7: Mission graph displaying state to represent progress.

Static mission graphs cannot display changes to the states of the nodes. However, in a digital version of mission graphs this can be done. For example, figure 5.7 displays a mission graph that does represent the state of the tasks: unavailable tasks have a gray outline and fill, available tasks have a solid outline and fill, and completed tasks are checked. Figure 5.7 represents a mission state where a player has progressed through half the entire mission. A dynamic representation of mission graphs displaying states in this way can be developed into a useful design and analytics tool. This notion is explored further in section 5.8.

At a first glance, when a task has only one predecessor, it seems to make little difference for a mission whether that predecessor is a weak or strong prerequisite. However, as there is no need to execute and complete a task that is a weak prerequisite in order to proceed, players might simply ignore weak requirements. In the case when a fight is a weak prerequisite to finding a key, players might ignore the fight and simply pick up the key. Would the fight be a strong prerequisite for finding the key, for example because the enemy is actually carrying it, players are forced to fight.

Weak prerequisites are quite common in games. For example, the two-dimensional, vertical scrolling shooter game *STAR DEFENDER* launches a sequence of enemies against the player (see figure 5.8). In this type of game, players are usually not required to actually kill those enemies; they can simply ignore them and proceed to the end of the level without firing a single shot.⁶ The structure of a game like that might be represented as figure 5.9. In a way, this structure can be replaced by a wide branching mission (see figure 5.10). After all, none of the tasks in the mission is a strong prerequisite for another; the tasks might be executed in any order. The player might even choose to complete the goal immediately. Once players start the mission, all other tasks become available. However, in this case, the order in which players will encounter tasks is lost. This order might be important, as increasing difficulty, pacing and task variation are all considerations that impact mission design. Therefore, the mission structures in figures 5.9 and 5.10 are not the same and specifying the order in which players will (likely) encounter weak prerequisites does matter.

⁶Although this would make the game very, very difficult. In addition, the final enemy of a level usually is a level-boss that must be defeated to finish the mission.

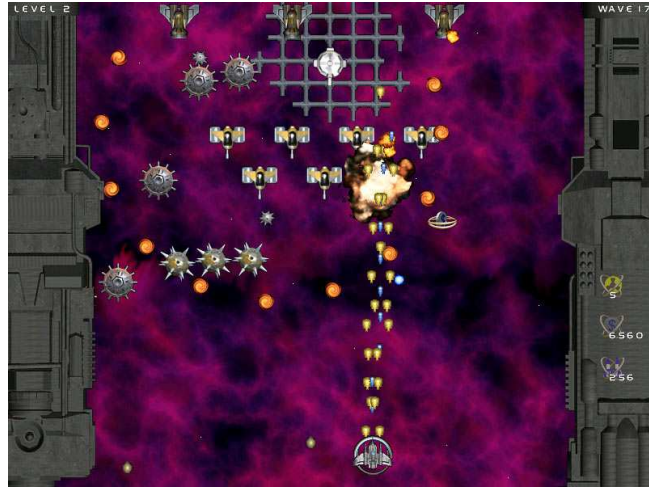


Figure 5.8: Many of the tasks in STAR DEFENDER are not required to finish a level.

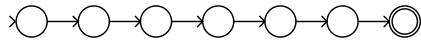


Figure 5.9: A linear mission with weak requirements.

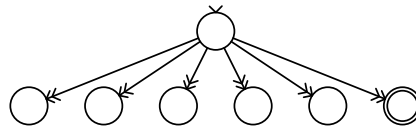


Figure 5.10: A branching mission with strong requirements.



Figure 5.11: A linear mission with one solution.

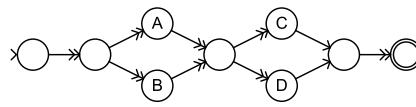


Figure 5.12: A linear mission with six solutions.

In addition, there is an important difference in the way strong and weak requirements map to game spaces. It makes sense to place all tasks with a weak prerequisite closely after that prerequisite. That way the order in which the player encounters them is preserved. On the other hand, because strong requirements dictate the order in which players can execute tasks more strictly, a task can be placed much further from a strong prerequisite. One might think of a weak prerequisite as a task players must encounter in order to get to the task that requires it. This difference will be explored in more detail later in this chapter and in chapter 7.⁷

Games with a linear structure of many strong requirements restrict the number of play trajectories considerably. For example, the mission in figure 5.11 only has one solution: players must simply execute all tasks in order to finish it. Although structures like this can be found in games, they offer little variation and replay value. Many games would do well to include weak requirements and branches of strong requirements in order to create more interesting missions. For example, the mission in figure 5.12 does not dictate in what order the player must perform tasks A and B, but both must be completed before the player can proceed. In addition, the player needs to perform at least C or D to be able to complete the mission. Even old and linear games such as SUPER MARIO BROS. often offered alternative paths with different challenges to get to the end of a level. Frequently, their mission structure is more like the one in figure 5.12 than the one in figure 5.11.

5.5 Space Graphs

In the Mission/Space framework space is also represented as graphs consisting of nodes and edges. In contrast to mission graphs, space graphs represent space directly, most nodes represent places the player can be in. A space graph has three different types of nodes. Any node in a space graph can be further specified by using colors and letters to indicate different types. The three types of space nodes are (also see figure 5.13):

1. A *place* is the elemental unit of space. What constitutes a place can vary from game to game. In a game with discrete spaces such as a text adventure or a tile-based board game, places correspond with the positions the player can occupy. In games with continuous spaces a place might be a room, a platform or a zone. Usually mechanics help define places in these games: if a particular platform gives the player different options to travel onwards than another platform in a gravity based platform game, then both platforms are considered to be individual places. Places are represented as open circles, whose size can be used to indicate the place's

⁷It might also suggest a second type of strong requirement, one that suggests a prerequisite must be completed and must be immediately precede the task that requires it in space. However, the precise nature of the tasks can be used to determine how far apart a task and its prerequisite can be placed: if the prerequisite is finding a key, they can be placed far apart; when the prerequisite is jumping across a wide gap, it must be placed immediately before the task.

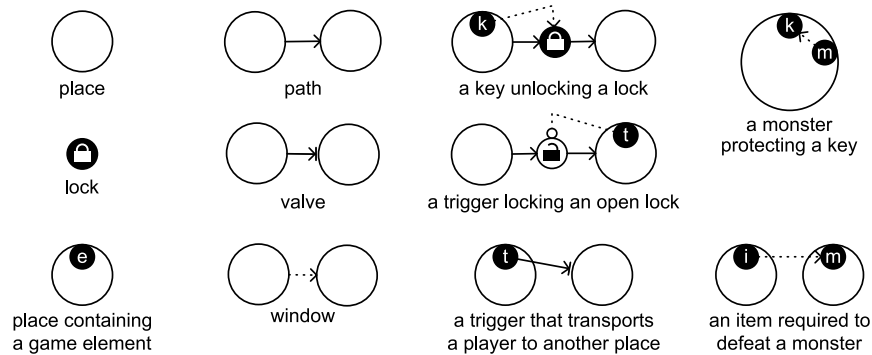


Figure 5.13: The basic nodes and edges of a Space Graph

relative dimension.

2. A *lock* is a special type of place that is not accessible until the player has activated or acquired the correct key or keys. A lock can be in two states: locked and unlocked. When a lock is locked it is represented as a black circle with a white lock symbol, when it is unlocked it is represented as a white circle with a black outline and a black opened lock symbol.
3. A *game element* is an item, character, creature or feature that can be found in a particular place. Game elements are represented as small colored shapes that are embedded within the circle of the place they are located in. Game elements often have letters indicating their particular type. Game elements can only be located in places; locks cannot contain game elements. Typical game elements that are frequently found in the examples that follow are entrances, goals, keys and triggers (marked with ‘e’, ‘g’, ‘k’ and ‘t’, respectively). These elements represent game objects that implement mission logic.

Space graphs can have many different relations represented by different types of edges. Some of these depend on a game’s particular implementation. The more common relations include:

1. A *path* indicates that players can move freely between two places. A path can be traversed in two directions, but as it is usually important to know which place the player is likely to encounter first, a path is represented as an arrow pointing towards the space that is further away from the player’s point of entrance.⁸
2. A *valve* connects two places or a game element and a place. It can be traversed only in one direction. A valve is represented as a solid arrow

⁸In addition, this difference will play an important role when performing transformations on game spaces, certain transformations will allow game spaces to be reorganized in such way that game element or connections are moved towards the starting location, but not the other way round.



Figure 5.14: A window in PRINCE OF PERSIA showing a potion that cannot be reached from this direction, prompting the player to paths that are not yet discovered.

with a different arrow head (\rightarrow) indicating the direction of travel. A valve between two places is only traversed when a player chooses to do so. A valve that starts from a game element automatically transports a player when the player activates the element (voluntarily or involuntarily). This can be used to represent teleporters or traps.

3. A *window* indicates that the player can see a place from another place, but cannot directly travel between these places. A window is not necessarily literally a window: any place the player can see but not immediately reach is considered to be connected by a window. Windows can prepare the player for challenges to come. They can also point the player at areas that are otherwise hidden or difficult to reach and the rewards that getting there might yield (see figure 5.14). In general, players assume that a visible yet inaccessible place does have a path leading to it. Windows are represented as dotted arrows connecting two places. The arrow points towards the place that can be observed.
4. An *unlock* relation connects a game element with a lock, or it connects two game elements. When connected to a lock, the source game element acts as a key to open it. Locks can have multiple keys in which case all keys must be acquired or activated before the player gains access to the lock.⁹ An unlock between two game elements indicates that the first element is required to use, obtain or activate the second element, or it might indicate that an element is protected by another element. Unlock relations are represented as dotted arrows pointing towards the lock or the element being activated.
5. A *lock* relation connects a game element with a lock. The game element acts as a trigger to close the lock. Lock relations are represented as dotted lines that end in an open circle at the side of the lock.

As with mission graphs, the basic elements of space graphs can be used to create a number of elementary constructions. These include, but are not

⁹Although it is imaginable that for a particular game this is implemented differently and any one key provides access to a lock.

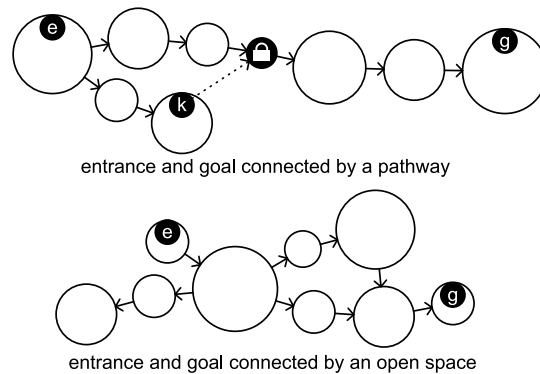


Figure 5.15: A pathway and an open space.

restricted to:

- A graph that connects two places with paths and valves is called a *pathway*. The pathway includes the two places it connects, and any places in between (see figure 5.15). The shortest possible pathway simply connects two places directly with a single path.
- A pathway that does not contain any closed locks or valves is called an *open space* (see figure 5.15). How the open space is constructed does not really matter; even if the place is constructed from a set of connected rooms in a textual adventure it is still considered to be an open space, as long as the player has the ability to travel to all places in the open space freely without having to activate game elements. A linear pathway through which the player can freely travel back and forth is also considered an open space.
- A *locked short-cut* can be used as an alternative for a valve. It blocks player's access from one direction but can be easily unlocked from another direction. Once it is unlocked it will stay open. Locked short-cuts are often implemented with doors that open only from one side (see example a in figure 5.16).
- A *hub* is a central place frequently visited by the player from which several separate (usually four or more) pathways start (see example b in figure 5.16). Although some pathways might be initially closed and some pathways might lead back to the hub through a valve. A hub is a good place to locate entrances and 'save points' as it can minimize the amount of backtracking after a set-back.
- A *set-back* forces players back to a previously visited place as a result of a failed challenge or through the use of traps (see example c in figure 5.16). It will take time and perhaps resources to return to an earlier place. Games with 'save points', and in which the player frequently dies, automatically

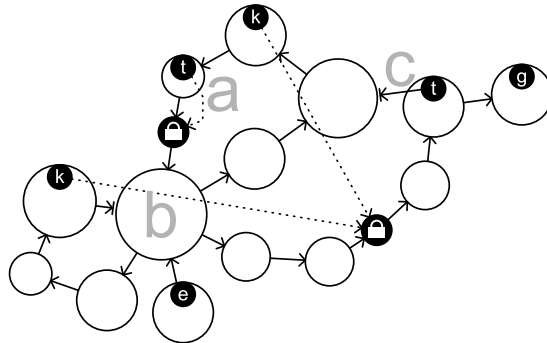


Figure 5.16: Example illustrating a locked short-cut (a), a hub (b) and a set-back (c).

have set-backs. Set-backs are also frequent in platform games where failing crucial jumps can force the players to lose much altitude that might not be easily regained (also see Compton & Mateas, 2006).

- In contrast to the set-back, a *force-ahead* pushes players into previously unexplored areas for which they might be ill-prepared or lack the proper equipment. In general, chances of player death and failure increase, but the player should be allowed a chance to return to the previous stage, leading to a survival challenge. The old dungeon crawler *EYE OF THE BEHOLDER II* includes trapdoors that drops the player into lower, more difficult part of the dungeon from which the player would struggle to return. A force-ahead can be realized with a valve to a new and very dangerous place.

5.6 Level Analysis: The Forest Temple

The differences between mission and space, and their relation, become more clear from the analysis of the ‘Forest Temple’ level in *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* (see figure 5.17). The *THE LEGEND OF ZELDA* series of games are well known for their quality game and level design and this game is no exception. In this level, the player, controlling the game’s main character Link, sets out to rescue monkeys from an evil presence that has infested an old temple in the forest. The mission consists of the player freeing a total of eight monkeys, the defeat of the mini-boss (the misguided monkey king Ook), finding and mastering of the ‘gale boomerang’ before finally defeating the level-boss (the ‘Twilit Parasite Diabara’). Figure 5.18 displays the forest temple level map as it is published in the official game guide. Figure 5.19 presents a mission graph and a space graph of the level. In order to reach the goal, Link needs to confront the level-boss in a final fight. In order to get to that fight, Link needs to find a key and he needs to rescue four monkeys, for which he needs the gale boomerang, for which he needs to defeat the monkey king, etcetera. Some tasks can be executed in different order: it does not really matter in what order Link liberates the



Figure 5.17: Confronting the misguided monkey king Ook to obtain the gale boomerang in THE LEGEND OF ZELDA: TWILIGHT PRINCESS.



Figure 5.18: The map of the Forest Temple from the official game guide (Hodgson & Stratton, 2006, 68).

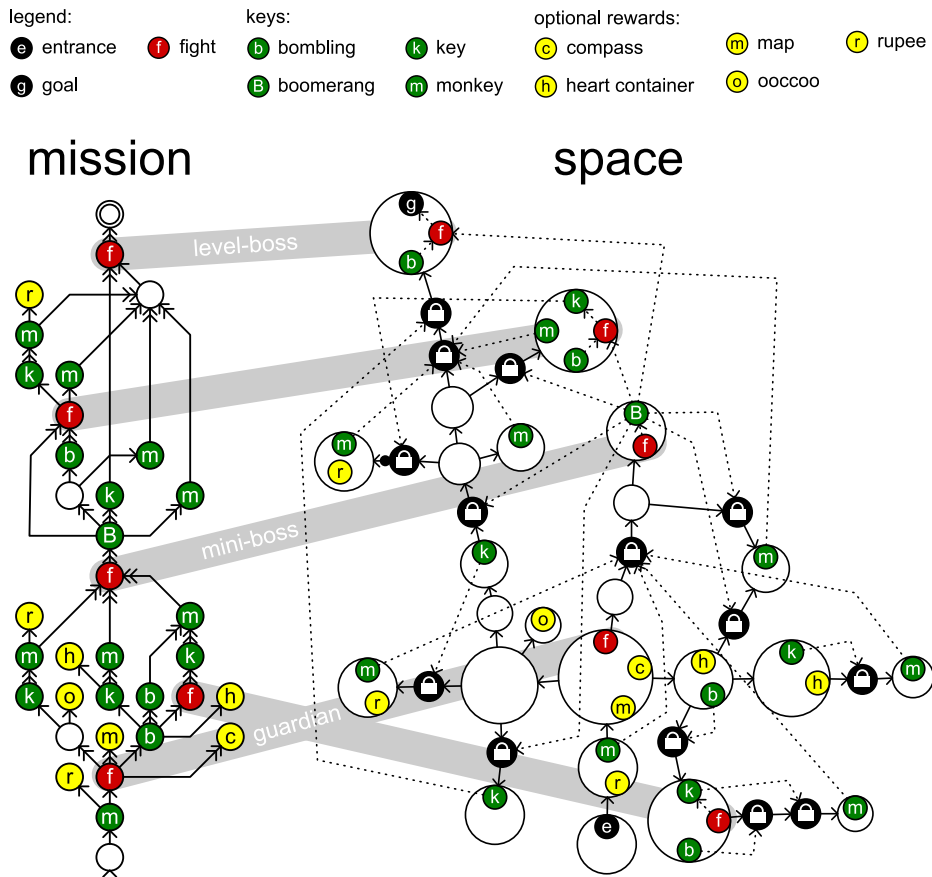


Figure 5.19: Mission graph and space graph for the “Forest Temple” level from THE LEGEND OF ZELDA: TWILIGHT PRINCESS. The gray lines connect the corresponding fights in both graphs.

monkeys two, three four. Other tasks are optional, but lead to useful rewards.

The mission structure for the Forest Temple level has a few striking features. One is the bottleneck formed by the fight with the mini-boss and the retrieval of the boomerang halfway through the structure and the two sections of relative nonlinearity before and after the bottleneck. This structure corresponds directly to the learning curve of the level in which the player needs to train with the new weapon before using it to defeat the level-boss. It is a structure that is frequently found in Nintendo games.

The game space features a hub-and-spoke layout (see section 5.1) that supports the parallel tasks of the mission structure. From the central hall (where the first guardian is fought) the player can go into three directions. The pathway that leads to the right, quickly branches into three more pathways. Three pathways lead to captured monkeys and one to the mini-boss. The last pathway is only open to Link after he has freed the first four monkeys. After the player has retrieved the boomerang, additional spaces in the first hub-and-spoke structure and a new hub can be reached. This structure is not the only possible space to accommodate the mission. It is not very hard to imagine different types of spaces that accommodate the same mission structure. Even a linear layout, in which the rooms are aligned in a long row with different tasks in each of them would be a possibility.

When playing the game, a trajectory through the space is generated by the player. This trajectory can be represented as a linear string of visited nodes. Trajectories that lead to the end goal of the level might be called solutions. The individual trajectory is constrained by the structure of the map. The player has to start at the entrance and the only way to progress is to find the first monkey. From there on an infinite set of trajectories becomes possible (infinite, because the space allows the player to travel back to earlier locations). Yet, this does not mean that all combinations are possible. Players must first defeat the mini-boss before retrieving the boomerang, and they must have retrieved the boomerang before they can reach the master key. The layout of space constrains the possible play-trajectories and solutions.

From this analysis it should become clear that both mission and space have their own individual structure. There is some natural affinity between the hub-and-spoke space and the required parallel branches in the mission, but the two are not directly linked. As mentioned above the same mission structure could be mapped to a linear structure in space, whereas a linear mission could also be mapped to a hub-and-spoke layout (one where the order of the spokes to be visited is fixed).

The gale boomerang itself is a good example of the lock and key mechanisms typical for the series and indeed this is used in many action-adventure games (cf. Ashmore & Nietzsche, 2007). Lock and key mechanisms are one way to translate strong prerequisites in a mission into spatial constructions that enforce that relation between tasks. The boomerang is both a weapon and a key that can be used in different ways. It has the capability to activate switches operated by wind. Link needs to operate these switches to control a few turning bridges to give him access to new areas. In order to get to the master key that unlocks

the door to the final room with the level-boss, he needs to use the boomerang to activate four switches in the correct order. At the same time the boomerang can be used to collect distant objects (it has the power to pick up small items and creatures), and can be used as a weapon. This allows the designer to place elements of the second half of the mission (after the mini-boss has been defeated) in the same space that is used for the first half of the mission. This means players will initially run into obstacles they cannot overcome until they have found the right 'key'.

5.7 Mission-Space Morphology

The structuralist tradition in narratology understands the part of the art of a well-told story in terms of the structural relations between the different elements in plot and narration. In a well-told story plot and narration are rarely the same. Narration speeds up through, or omits uninteresting events. Narrations that use flashbacks or flash-forwards present the events in the plot in a different order than they have occurred. A common trick is to start a narration “in medias res”, where the narration starts in the middle of or just the main action, only to step back to relate the events that led up to the starting scene. Differences between plot and narration are powerful narrative devices that are exploited by master storytellers in any medium. What is more, a plot is not necessarily linear, it might contain events that occurred in simultaneously; when the story is narrated in a linear medium (as most traditional media for storytelling are), the storyteller must use these devices to be able to tell the story at all. Traditional structures and templates exist that have proven their value in the past and are still used in books and films alike. From the structuralist analysis of Russian fairy tales by Propp (1968) and theory of the monomyth of Campbell (1947) to more modern interpretations used in contemporary cinema by Vogler (2007). Many of these structures facilitate anticipation and involvement of the reader with the narrated events. Differences between plot and narration are employed to foreshadow, to change perspectives, to shock and to educate. The possibilities are endless.

Similar, commonly used structures do exist for game spaces and missions. In fact, the structures in figure 5.1 are examples of this type of structures, even though mission and space are somewhat tangled up in them. Currently much level design depends on a direct mapping between mission and space: all too often, mission and space are isomorphic structures. A strong indicator for this dependency is the popularity of the quest in many adventure games, a genre that typically features strongly articulated spaces and missions. In the trope of the quest, in which the journey of the hero indicates personal growth, mission and space are isomorphic; they share the same structure. In many ways this is very convenient. DUNGEONS & DRAGONS designer Dave Arneson is attributed to have once remarked that DUNGEONS & DRAGONS levels (or dungeons) are intentionally designed as flowcharts.¹⁰ It could be argued that a certain level of isomorphism between mission and space guarantees a smooth player experience.

¹⁰From personal correspondence with David Ethan Kennerly (2009).

In its most extreme case where a player is both railroaded through space and mission the play experience can be tailored to that one single trajectory. *HALF-LIFE* and *HALF-LIFE 2* have refined this strategy to perfection, and to great commercial and critical success (see section 1.6). Strong isomorphism between space and mission allows the use of many spatial metaphors or other spatial narrative devices. The symmetrical pathways mentioned above, could for example be used to suggest a similarity between the tasks that must be completed along those pathways.

On the other hand, in the ‘Forest Temple’ level, mission and space are not isomorphic (see figure 5.19 above). During the first half of the level, the mission with its parallel chains maps fairly directly to the hub-and-spoke layout of the space. But once the gale boomerang is obtained the player has to traverse back through the dungeon to gain access to previously inaccessible places. Would mission and space be more isomorphic, Link would have to travel further into a new section of the dungeon. Of course, as conventional level design wisdom dictates, there is little point in using a lock and key system where Link encounters all the keys first. In general, it is better to have the player find the lock before the key in this way for three reasons.

1. When keys are encountered first, players will simply be forced to collect everything they encounter without discrimination, which makes rather simplistic gameplay.
2. With obstacles and items that act as locks and keys but are represented as something else, it is easier to recognize the key if players know what the lock is; players then usually realize where they can proceed; they will actively formulate the intention to return to the lock.
3. When players can negotiate obstacles they were unable to get past earlier, they will experience progress and accomplishment. Thus, forcing Link go back to previously visited places foregrounds his growth as a character. Where there were obstacles he could first not overcome, he now can conquer that space. It sets up a natural, clever contrast between Link’s state before and after the defeat of the mini-boss and marks his progress along the hero’s journey.

There is no reason why in a game mission and space should be isomorphic. In fact, just as stories benefit from different morphologies for plot and narration, so do games benefit from having different morphologies for mission and space.¹¹

5.8 A Software Tool for Level Design

As is the case for the Machinations framework, the Mission/Space framework is formal enough to be implemented as an automated design tool. A tool like

¹¹On a side note, I once experimented with an ‘in medias res’ structure for a tabletop role-playing session, where I had players play through the events that lead up to the opening scene they played through earlier. ‘In medias res’ is also used in *PRINCE OF PERSIA: SANDS OF TIME* where most of the game is framed as a long flashback. In both cases the structure translates well to games and actually contributes positively to the gameplay experience.

that, with the name *Ludoscope*, was created as part of this research. Ludoscope allows a designer to draw mission and space graphs efficiently. In fact, the mission and space graphs that feature in this chapter were created with Ludoscope. However, Ludoscope is more than just a drawing aid. It also implements the logic that governs mission and spaces. As a result, it can be used to simulate levels at an early stage and help developers identify design flaws.

The implementation of mission graphs in Ludoscope allows designers to hook up a network of tasks. Designers can indicate which tasks are entrances and which tasks are goals. Based on that information, it determines which tasks need never be encountered and marks them as such. By selecting a task, the designer can immediately see what other tasks can be reached from that point in the graph, these tasks are given a green outline. When a selected task inhibits another task it displays the inhibited task with a red outline. Other tasks that have become unreachable because of this are also displayed with a red outline (see figure 5.20). This allows designers to select tasks one at a time in simulation of a player completing tasks in the game and checking if the mission still can be completed. Ludoscope can easily be extended to include more checks for inconsistencies or warn the designer for potential deadlocks. In addition many types of heuristics for games might be implemented to assist designers, such as space syntax measures described by Paul Martin (2011).

The implementation of space graphs in Ludoscope allows designers to do something similar with spaces representing missions as well. By selecting a place in a space graph, all places that can be reached from that location are rendered in with a green outline. In addition, designers can select game elements to unlock (or lock) locks. Again, this allows them to simulate a player's progress through a level (see figure 5.21).

Ludoscope is still work in progress, and as will be discussed in Chapter 7 has much more functionality than drawing Mission and Space Graphs only. In particular, Ludoscope initially evolved from prototypes for procedural level generation tools, and it contains support to represent and involve mechanics through Machination diagrams as well. This latter aspect allows designers to zoom in on the relation between their game's mechanics and level design, which, as will be argued in the next chapter, is a critical aspect of designing games where

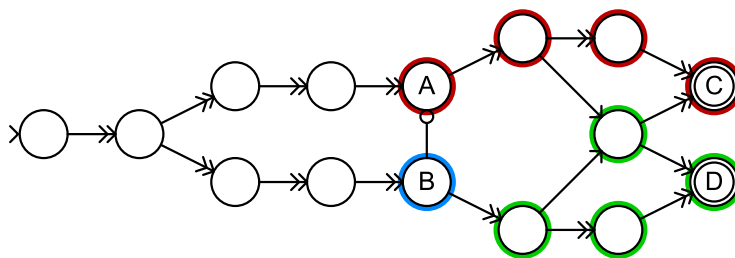


Figure 5.20: Selecting task B in Ludoscope reveals that task A is inhibited making it impossible to reach goal C. Goal D can still be reached.

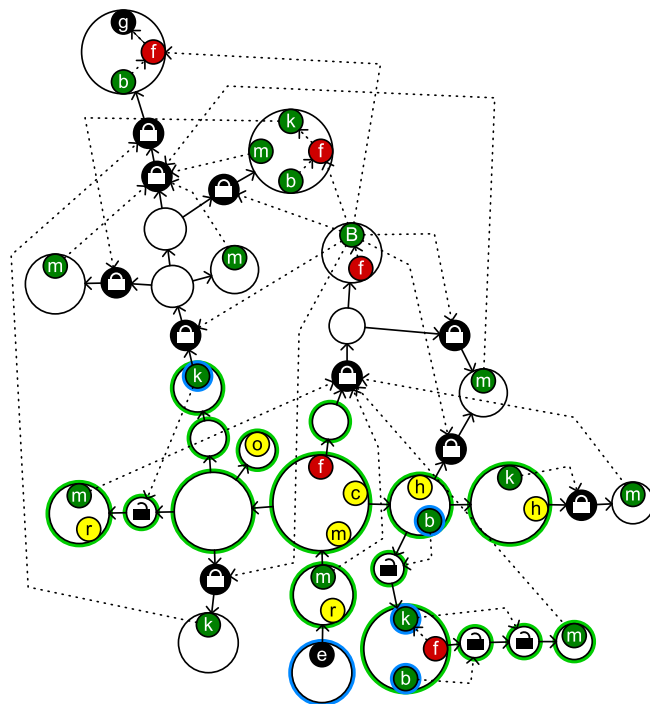


Figure 5.21: Simulating the Forest Temple level. The light blue outline indicates selected places and activated game elements. By selecting the place containing the entrance and various keys, player progress can be simulated. In this case, the simulated player is in the early stages of the level having made some progress in the lower left and right corners.

structures of emergence and progression are tightly integrated.

5.9 Conclusions

As is the case with game mechanics, level design benefits from a more structural approach. A game level is a complicated construction consisting of many different parts. Different models and perspectives can help level designers to keep their focus and to structure their labor. To this end, this chapter introduced the Mission/Space framework. This framework includes specialized graphs to represent missions and spaces separately.

Mission graphs focus on the tasks and challenges that players need to complete in order to finish a level. There are three types of relations between the different tasks in a mission: weak requirements, strong requirements and inhibition. Mission structure foregrounds the interdependencies between tasks and can be used to identify potential deadlocks or other inconsistencies, as well as a balanced learning curve and training structure. Space graphs represent the topographical structure of the level. They are indicative of how different places are connected, a level's flow, pacing, and what game elements can be used to unlock new areas.

Mission and space are related, creating spaces to accommodate missions or creating missions to suit a particular space is an important aspect of level design. Although, sometimes games have levels where missions and spaces are almost isomorphic, the structures of mission and space are independent. A detailed analysis of the Forest Temple level in *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* showed that games benefit from more sophisticated relation between missions and spaces.

In order to aid level designers, the Mission/Space framework can be used to create level design tools that help designers create complex but consistent levels. The Ludoscope prototype implements some of these ideas. It allows designers to create mission and space graphs and subsequently simulate players progress through them.

The Mission/Space framework provides a solid structure that can be used as a stepping stone in the development of further theory for game design. In the next chapter, the Machinations and Mission/Space framework will be used to explore how structures of emergence and progression can be integrated. In Chapter 7 both frameworks are leveraged to create a framework and prototype that allows the automation of certain task in order to support the creative process of designing games.

The challenge for game designers who want to create rich, open game worlds and tell interesting stories at the same time, is to move beyond the constraints of unicursal corridors or multicursal hub structures while keeping the player's attention on a storyline. And it is no easy task.

Espen J. Aarseth (2005, 11)



Integrating Progression and Emergence¹

This chapter focuses on the relation between mechanics and levels, which were the subjects of the previous two chapters. The Machinations framework and the Mission/Space framework formalize different perspectives on these two aspects of games. Machinations diagrams help visualize and understand a game's internal economy. It focuses on the mechanics that govern the economy and contribute to emergence in games. The Mission/Space framework, on the other hand, focuses on the construction of levels, and the mechanics that control players progress through a game. These two aspects of game design: mechanics and emergence, versus level design and progression, are often understood as two opposing or conflicting ways of generating gameplay. It is the goal of this chapter to leverage both frameworks in order to integrate structures of emergence and progression more closely and bridge the gap between game mechanics and level design. Ultimately, a close integration between emergence and progression would lead to games where progression through levels, and perhaps stories, emerges from the rules.

In section 6.1 I will discuss how professional designers typically create mechanics before they start focus on level design. In the two sections that follow, I will explore how the relation between mechanics and levels is typically approached in games that traditionally rely on structures of progression, and sandbox games that allow players to build and structure the internal economy themselves, respectively.

Integrating levels and mechanics is not easy, and the number of games that have successfully done so in the past is low. In section 6.4 I will discuss the limitations that have been encountered in designing game levels that display more dynamic and adaptive behavior. The problem often resides in a conceptual mismatch between relatively crude mechanisms for level and story on the one hand and quite sophisticated and expressive mechanisms for physics simulations and/or economy on the other. In section 6.5 I will explore how feedback mechanisms might be applied to mechanics to control progression. This explo-

¹This chapter is an extended version of a paper presented at the DiGRA Conference in 2011 (Dormans, 2011b).

ration will lead to two new techniques to create progression through emergence: 1) feedback mechanisms for lock and keys, and 2) modeling progress as an economic resource. These techniques are discussed and illustrated in the final two sections of this chapter.

6.1 From Toys to Playgrounds

In general, mechanics are designed before levels, and this means that in most games, level design is dependent on the game mechanics. Particular mechanics often serve as a starting point for levels in a game: a level might introduce or focus on particular mechanics. For this reason, it is usually not possible to finalize a game's level design before finalizing the mechanics. After all, the challenge posed by a platform carefully placed at a distance that is just less than the maximum jump distance completely changes when the jumping distance is increased. Worse, what was intended as an impossible jump, might change into a possible jump or the other way round. It is also one of the reasons why the MDA-framework stresses that game designers look at games starting with rules and mechanics (see section 3.2).

Kyle Gabbler summarizes the dependency of level design on game mechanics as one of seven tips in his video keynote for the first Global Game Jam in 2009 (Gabler, 2009).² He advises game designers to “make the toy first”: he urges the participants not to create any art assets until the basic mechanics of the game are in place. These basic mechanics should already deliver an interaction that is fun, even without goals or well designed levels. For example, if one sets out to build a racing game, the car should be built first, it is only when the car is fun to drive around, that one should start work on graphics and on tracks. From my own experience as a Global Game Jam participant in 2009, 2010 and 2011 I can testify that this tip is very useful.

Gabler's tips are not restricted to game jams and games built under extreme time constraints only: they are valuable for any game development project. Games are developed by creating many prototypes during short iterative steps. The earlier prototypes tend to resemble Gabler's ‘toys’: they often implement the basic mechanics and interactions, without spending much attention to goals or levels. In most cases, the levels that exist at this stage, serve as testing grounds to illustrate the basic game mechanics. Designers typically throw in as many game objects as they can for play-testers to play around with. The goal of these prototypes usually is to explore the limits of the mechanics, and to look for opportunities to create interesting interactions, or explore how game objects might be combined into interesting puzzles and challenges. It is only when the designers are convinced that the basic mechanics are solid that the virtual space is organized into a level set up to deliver a concise and well-structured experience; when the individual toys are organized into a playground.

The emergent simulation game *SPORE* was developed using a large number of small prototypes, and serves as an excellent illustration of this process. What

²The Global Game Jam is an annual event in which teams at hundreds of locations all over the world enter a competition to build a game based on a set theme within 48 hours.

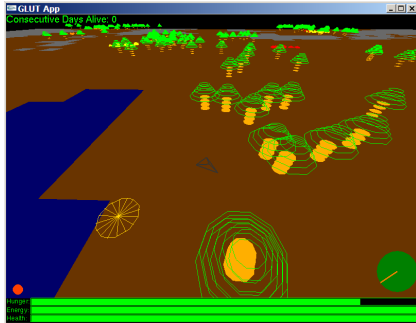


Figure 6.1: SPUG prototype for the creature stage in SPORE



Figure 6.2: Screenshot from the creature stage of SPORE.

is unique for a large commercial game, such as SPORE, is that many of these prototypes are available for download.³ These prototypes were built for various reasons. Some were experiments with the procedural techniques used to create terrains, worlds and even entire galaxies. While others are gameplay experiments designed to try out various rules and behaviors. One of these, SPUG (figure 6.1), is a prototype for the “creature stage” in the published game (figure 6.2). According to accompanying text on the website in this prototype “no limitations are placed on leveling up or cheating stats. This tool was intended to give designers the opportunity to explore different economies for the creature game, so limitations on power-ups and level ups are self-imposed.”⁴ Thus it was used to experiment with the same type of game mechanics that is the focus of the Machinations framework.

The design strategy that gives mechanics dominance over level design has a number of implications. It provides certain types of games with a natural structure for progression, as will be explored in the next section. Other games, that focus more on dynamic gameplay created by sophisticated mechanics (as explored in section 6.3), use levels only to set some rough boundaries that provide some color and perhaps some constraints on the emergent gameplay.

6.2 Progression through Structured Learning Curve

One of the many considerations of designing levels, is to train the player in the required gameplay skills necessary to complete the game. In general, players do not want to read manuals in order to play a game; they expect that learning the mechanics is a natural result of playing. This means that levels need to be structured in such ways that players actually learn the game while playing it. This type of structure fits well with games of progression that frequently model their structure after the hero’s journey. A sense of growth and accomplishment drives these games and sets up a particular relation between game mechanics

³See <http://eu.spore.com/extra/?id=10488> (last visited July 17, 2011).

⁴See <http://eu.spore.com/extra/?id=10626&lang=en> (last visited July 17, 2011).

and level design that is common among action-adventure games, first-person shooters and genres that typically include structures of progression.

Daniel Cook's skill atoms constitute one of the most concrete theoretical perspectives on this aspect of level design (Cook, 2007). He analyses the individual steps a player goes through in learning a new game skill, and the way individual skills are hooked up into chains. Once the design team has decided on the final mechanics to be included in the game, levels can be structured in such way that the player is taught these mechanics.⁵ The most straightforward approach is to spread out the chain of skills over the level and to organize the level accordingly. In this case, the chain of skills is integrated into the level's mission structure, which is then related to the level's spatial layout in a similar way as described in the previous chapter. However, levels are not there to teach the player the required skills only; there is usually more to a level than just a tutorial. Levels are also structured to facilitate exploration. Once the player has learned the basics of playing a particular game, levels provide the player with opportunities to display their mastery. During this stage, the mechanics become a means towards the goal of exploring the level or completing an interactive story.

The way the player is taught to use new items and strategies through level design also resembles the learning stages found in martial arts training (Kohler, 2005; Dormans, 2005). This structure plays an important role in the level's mission and is a common feature in many action games, in particular action games produced by Nintendo. It is an excellent technique to learn the player the required actions to finish the game. It is an easily recognizable pattern of a game's mission, although to use it to the best effects, the space needs to be designed to support this structure. In both action games and martial arts, students need to master a whole vocabulary of different moves and combinations of moves. In martial arts learning progresses through four stages in which techniques are learned, practiced, combined and tested. These stages, called *kihon*, *kihon-kata*, *kata* and *kumite*, are all present in the Forest Temple level of THE LEGEND OF ZELDA: TWILIGHT PRINCESS (see section 5.6). The *kihon* stage, in which a player learns an individual technique in isolation and relative safety, is present in both the earlier tasks involving the 'bomblings' and the tasks for which the player needs the 'gale boomerang'. These tasks are repeated a few times (*kihon-kata*) before the player is taught how to use the boomerang to pick up the bomblings to deliver them over greater distances before he can reach the final two monkeys (*kata*). Finally the player needs this combined technique in order to defeat the level-boss (*kumite*). The order and logic of these stages is dictated by the mission structure. The player cannot encounter the level-boss before she successfully demonstrated the techniques needed to defeat it (see figure 6.3).

This structure of introducing game mechanics gradually and having them act as locks and keys can be found in a very pure form in certain smaller independent

⁵Although I like to point out that the development process for games is highly iterative, and that work on level design is often started before all mechanics are complete. It is not uncommon that mechanics are changed because of insights gained during the design of levels. However, the design process usually starts with mechanics, and mechanics are usually made definitive before all levels are designed.

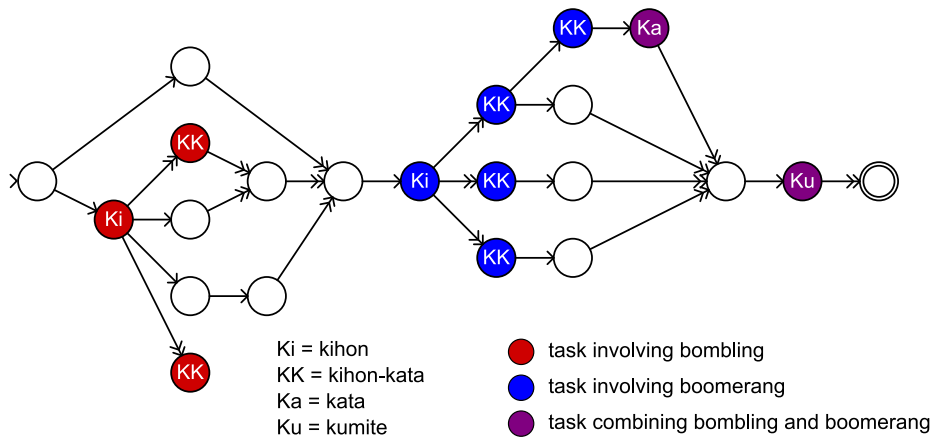


Figure 6.3: A simplified version of Forest Temple mission highlighting the kihon, kihon-kata, kata, kumite structure.

games. *KNYTT STORIES* and *ROBOT WANTS KITTY* are good examples of such games. Both of these games are platform games where the player's goal is to reach a particular location (even though these games' story might frame it a little bit differently). Both basically consist of one large level where the player gathers a number of power-ups that act as locks and keys. But also the challenges the player meets get progressively more difficult. Where, for example, the double jumping ability allows the player to jump longer distances in both games, the gaps the player needs to jump across do get wider, and the penalty for failing a jump increases from, having to replay a little part of the level in order to try again, to dying and/or replaying longer parts of the level.⁶ Table 6.1 lists the power-ups in *KNYTT STORIES* in the same order they are encountered. Note that some of these power-ups combine in even more powerful combinations. For example, once players have found both the double jumping and wall climbing abilities, they can use a double-jump after jumping from a wall. Likewise when players have the double jump and the high jump, they can use both to jump even further.

Larger games, such as *THE LEGEND OF ZELDA: TWILIGHT PRINCESS*, take more time to introduce their mechanics. As we have seen in the detailed analyses of this game's Forest Temple level in section 5.6, only two important mechanics/power-ups were introduced in this level: the 'gale boomerang' and the use of 'bombing' creatures. *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* typically takes more time to train the player in the use of these mechanics, and explores more possible combinations of the two, allowing it to design an entire level around these two mechanics. Although I estimate the average time a player

⁶A double jump is a common mechanism found in many platform games. A double jump allows player controlled characters to jump once while in mid-air, effectively allowing the player character to jump twice. It can be used to jump higher or further and emphasizes the timing involved in performing jumps.

Type	Description	Associated lock
Run	The player can now move faster.	Jump across wider gaps, it also becomes easier to jump past enemy creatures.
Climb	The player can now climb and jump from vertical surfaces.	Tall vertical structures too high to jump.
Detector	The player glows red when certain types of enemies are close. The player glows green when a secret passage is near.	None, this power up is optional (but very useful).
High Jump	The player can jump higher.	Wider gaps; creatures that jump low when the player jumps; jump and climb to higher locations.
Double Jump	The player can jump a second time in mid air before landing.	Wider gaps; creatures that jump low when the player jumps; jump and climb to higher locations.
Eye	The player can now see invisible creatures and platforms, which curiously did not affect the player before.	Previously invisible platforms allow the player to get to places that were impossible to get to earlier.
Umbrella	The player can use the umbrella to glide and even float up in areas that have upward drift. The Umbrella also protects from enemies that shoot 'bullets' from above.	Upward drift, extreme long jump (especially when combined with a high, double jump).
Hologram	Player leaves a holographic projection fooling enemy creatures.	Creatures that otherwise block your passage when the player gets close.

Table 6.1: The power-ups of KNYTT STORIES in the order they are encountered and their associated locks.

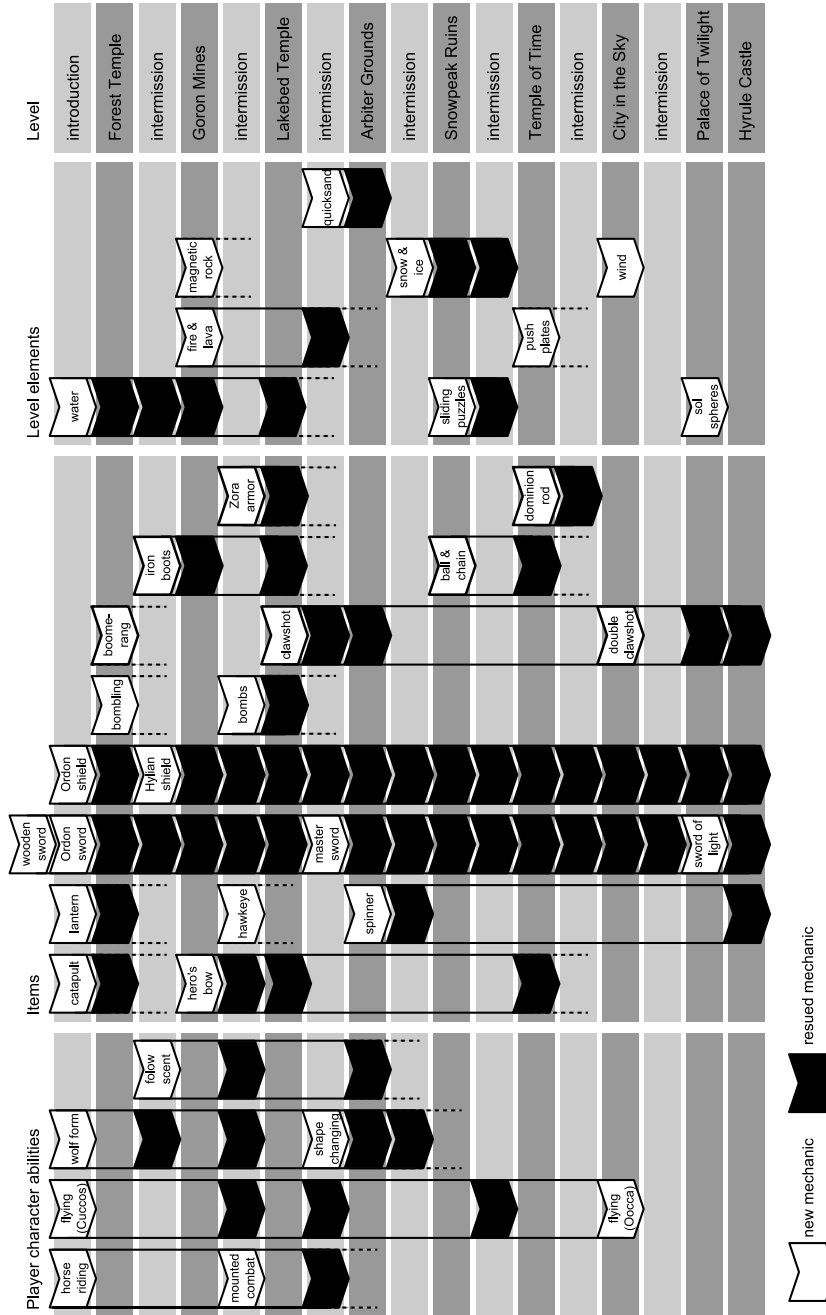


Figure 6.4: Game elements as they are introduced in THE LEGEND OF ZELDA: TWILIGHT PRINCESS. The data for this figure was collected from playing the game and consulting a printed game guide (Hodgson & Stratton, 2006).

will spend in the forest temple level is roughly half of the time spent to complete *KNYTT STORIES*, *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* still takes more time to introduce each power-up and explore its potential. As players explore more levels in *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* more and more mechanics are introduced, although every level focuses on only a few new mechanics and a few reused mechanics. Figure 6.4 provides an overview of this game's most important mechanics, when they are introduced and in what levels they are reused. Mechanics that only act as locks and keys (such as 'fused shadows' and 'mirror shards') without being weapons or affecting the gameplay are omitted in this figure. Many of the mechanics combine in interesting ways. For example, wearing the 'Zora armor' allows the player to breath under water, when also wearing the 'iron boots' will make the player sink allowing the exploration of deep water caverns. In addition, the enemies and traps players encounter require an increasing mastery of the game's mechanics and their combinations to defeat or avoid them.

What both *KNYTT STORIES* and *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* do well, is to integrate the mechanics that control the progression through the game world with the emergent mechanics that determine the core gameplay. This is an effective and proven way to combine emergence and progression in a game. This is testified by the success of the Zelda games and the frequency with which mechanics or power-ups are used in this way. However, whether the two modes are truly integrated remains doubtful. Progression in these games still relies on a fairly low number of game state changes, and needs to be planned in detail to work. Over the past few decades, game designers have learned to forge the two into pretty smooth game experiences. But it has not led to a true synthesis of emergent and progressive gameplay.

6.3 Economy Building Games

Strategy games and management simulation games usually follow a different design strategy to integrate their mechanics into level design. In stead of creating levels to dictate a structure of progression, these games allow players to build and shape an emergent, internal economy within the constraints set up by the level's space or mission. This design strategy can be found in computer games as *CIVILIZATION*, *CAESAR III* or *STARCRAFT*, but also for modern, management simulation board games such as *PUERTO RICO*, *CAYLUS* or *AGRICOLA*. In these games an internal economy is built during play. Game spaces constrain the way the economy might be built, but rarely accommodate a mission in the sense that was discussed earlier. Instead of traveling towards a particular goal, the goal of the game is to build up an effective game space. Missions, insofar they exist in these games, often constitute of the single task to build an economy according to some sort of specification. These economies usually start out quite simple with the production of basic resources, but tend to get more complicated quickly. For example in *CIVILIZATION* players build cities to produce food, wealth, knowledge, buildings and units. The location of the city affects the production rates: building a city on fertile grasslands will increase the food production, rivers will



Figure 6.5: In the game CAESAR III players build cities in the Roman era.

increase trade and wealth, while hills and mountains offer the opportunity to build mines to increase production of buildings and units. Players need to find locations that best suit their needs. A player that is going for a strong military will need more production, while building close to rivers can speed up trade, wealth and scientific advancements. There are both long term and short term considerations: cities that grow fast will eventually output more resources than cities that are located close to interesting resources but far from fertile soil. The default mode of playing CIVILIZATION generates new maps for every game the player starts. Players will thus have to make the most out of the land they have discovered.

Modeling a game like CIVILIZATION with the use of the Machinations framework presents us with a problem. Although many of the individual mechanics can be easily captured using machinations diagrams, different sessions require different diagrams as players effectively hook up game elements differently every time. By building and changing elements in the game's space, players also set up their own game economy. These economies usually start out quite simple with the production of basic resources, but they tend to quickly get more complicated. For example, in the Roman city simulation game CAESAR III (see figure 6.5) players build cities in the Roman era. Players need to build infrastructure for traffic and water, buildings to produce food and other basic resources. To build up the city's economy the player needs residences, workshops, markets and warehouses. Citizens will demand temples, schools and theaters, and at the same time the player must provide security against different types of threats by building prefectures, city walls and hospitals. Finally the player must train soldiers to protect the city from invading barbarians. The cities economy is dominated by a

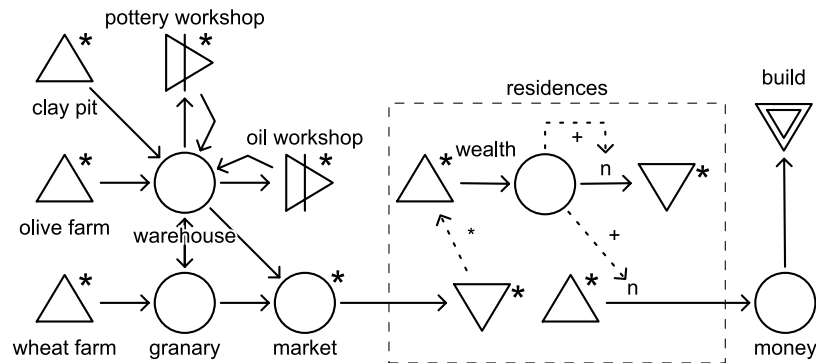


Figure 6.6: The basic economic relations between the elements in CAESAR III.

multitude of resources. Farms produce wheat, fruits, or olives, clay pits produce clay that can be converted into pottery in special workshops. Other workshops convert olives into oil, or metal into arms. The residences the players build are in constant need of these and other goods. The better the player can supply these residences the wealthier their inhabitants become, and that will in turn improve tax income needed to build more farms, workshops and residences or pay for other needs such as prefectures to fight crime and fire or military structure to protect the city from invading barbarians. In the meantime players will need to build granaries, markets and warehouses to distribute all these needs effectively over the growing city. Figure 6.6 represents the basic economic relations between some of these elements. The consumption of trade goods in residences triggers the production of wealth. More wealth has a positive effect on the amount of money generated through taxes. At the same time, wealth drains quickly creating an ever increasing need to supply residences with high quality trade goods. In the game the actual connections between all these elements are flexible: a farm might deliver its crops to a granary, warehouse or workshop depending on the needs and the distances to these locations (see figure 6.7). The challenge of CAESAR III is to utilize space effectively and build a smooth running economy. Players gradually build this economy as they see fit, but it will invariably be dominated by the positive feedback loop that involves production, consumption by citizens and tax income. This positive feedback is balanced by the negative feedback provided by the dynamic friction built into the citizens mechanism (see figure 6.8). The more effective players are in utilizing space and building up their city, the more effective their economic engine will run.⁷

Games where players build the game economy in this way, very clearly fall into

⁷CAESAR III includes a few more elements that I have left out for the sake of discussion. In the real game players also need to manage a number of hazards such as crime and fire by building special buildings to counter these effects. They serve to complicate the production mechanisms further. Apart from nutrition and wealth the citizens also demand entertainment, culture, education and religion which are produced and consumed in similar ways. Finally, in most levels of CAESAR III players also need to deal with demands of the Roman emperor and fight invading barbarians, they act as extra (periodic) friction to the economic engine.

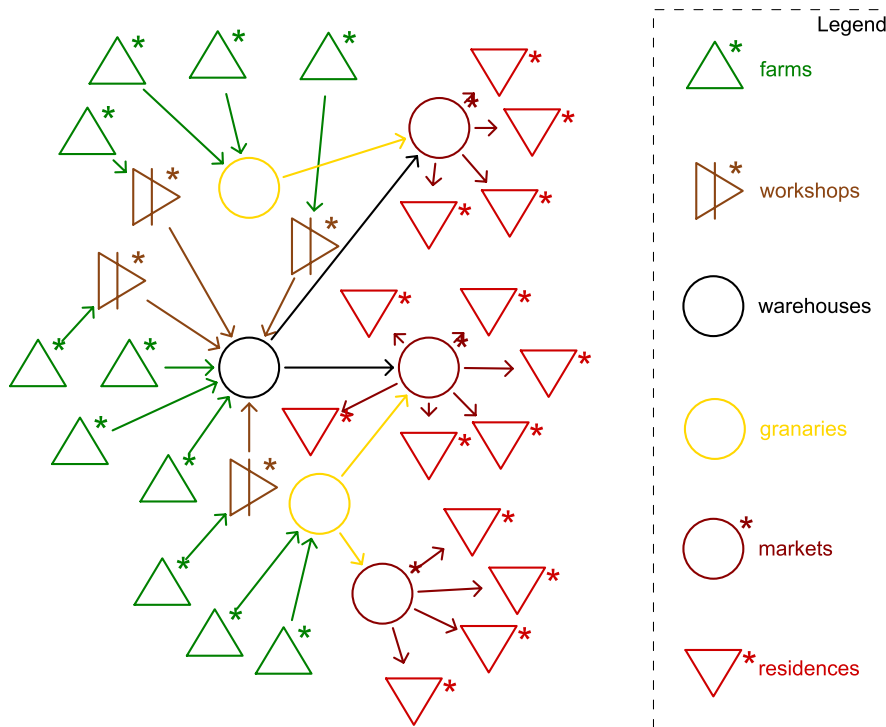


Figure 6.7: A possible spatial configuration for some elements of CAESAR III.

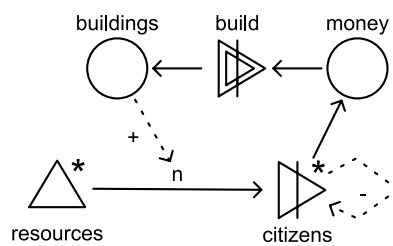


Figure 6.8: The dominant economic structure of CAESAR III.

Jesper Juul's category of games of emergence: it is almost impossible to write a walkthrough for *CIVILIZATION* or *CAESAR III*, but there are plenty of strategy guides available online. Still, playing these games does offer something of an experience of progression. *CAESAR III*, for example, has a series of scenarios, each with its own particular challenges and goals, and within each scenario there are a number of scripted events. But even without these events, the process of building a city goes through a number of stages, from the initial planning stage, when players still have much money to build anything they might need, to the managing of a crisis or the city's defense, to the tweaking and fine-tuning of the city's economy in order to reach very hard economic goals during the end stages of the later levels. *CAESAR III*, as many other games of emergence, has its own rhythm and progression that partly emerges from its dynamic game economy, and partly from the scripted events that are unique for every scenario.

To design mechanics and to plan scenarios for these games is not easy. It requires a thorough understanding of the game's potential economy. Most of the game elements the player can add to the game will somehow expand the game's economy, but there are only so many ways these elements can combine. Most of these games will have a dominant economic structure as the one presented in figure 6.8: these structures are the most effective way of setting up the economy and players will eventually veer towards building similar structures. In some cases, a game might contain a few alternative dominant structures. Most of the time players will be able to familiarize themselves with only a few elements at a time, learning the particularities of the game's potential economy over multiple sessions. This requires that players should be able to combine individual elements in different ways, and preferably in such ways that these combinations offer different strategic options.

At the same time, to keep players on their toes, a few scripted events, or a rhythm that is dictated by a mechanism that is not very emergent at all, can be used to structure scenarios and can give the game clear goals or closure. *CAESAR III*'s scripted events are one example of these, but similar mechanics can be found in many other games, including board games (where one perhaps least expected such mechanics). In the board game *CAYLUS*, for example, players compete for the favor of king Charlemagne by building him a castle and surrounding town. The game progresses through three stages after which a part of the castle is finished and the players are rewarded for their contribution (or punished for the lack thereof). Every round, the timeline that keeps track of the progress is advanced one or two steps based on the players' actions. Once the timeline is advanced at least six steps, the first stage is completed; after at least twelve steps the second stage is completed and after at least eighteen steps the third stage is completed and the game is finished. This timeline drives the game towards a conclusion, but does not constitute what I would call a dynamic mechanism.

6.4 A Mismatch Between Mission and Space

Although the design strategies to integrate mechanics and progression as described in the previous two sections are successful in their own way, it seems

that the development of design strategies that integrate the two more closely has hit a barrier. This barrier seems to come from an imbalance in the advances that have been made with creating sophisticated mechanics and game spaces on the one hand, and the lack thereof in the development of missions and interactive stories (Wardrip-Fruin, 2009). This mismatch between the mechanics of mission and space has held back further integration of emergence and progression.

Over the past few decades the game development community has accumulated much experience with creating compelling game spaces with interesting rules. From the early limited spaces from the seventies and eighties to the vast virtual areas found in modern games, game spaces have grown into highly detailed constructions with near analogue qualities. Traversing the space of a contemporary game is no trivial task, especially for those games that involve movement in their core mechanics, as is the case with most action games. But also for strategic games this evolution has been fast. One needs to simply compare the open free world of *STARCRAFT II* to the tile based combat found in *CIVILIZATION* or indeed classic boardgames such as *CHESS*, to appreciate the strategic depth allowed by freely positioned units and more continuous terrain features. Seeing these huge strides in the development of game spaces towards structures with a high granularity, I agree with Noah Wardrip-Fruin who observes that it is curious that game stories and quests have not grown as much; game missions usually work with a very limited set of possible states, all of which are known before play (Wardrip-Fruin, 2009, 59).

One cause for the mismatch in granularity between mission and space, Noah Wardrip-Fruin points out, is the popular quest logic and dialog trees common to most games. The player's progression through a mission is simply tracked by setting up a few bottlenecks or gates to act as milestones in a story. Once the player fulfilled the task associated with a milestone, the story is advanced. The implementation is as simple as keeping track of a few simple Boolean story flags that control the visible entries for the in-game journal that records the game story (Wardrip-Fruin, 2009).

For example, the story-driven first-person shooter game *DEUS EX* has detailed rules to simulate combat in urban environments, where players can use different strategies to overcome obstacles. An important aspect of the gameplay is that players can choose between various upgrades of their character, allowing them to specialize in stealth or direct combat, hacking computers or bypassing locks. Most of the levels are set up to accommodate different playing styles. It is even possible to complete the game without using lethal force against human opponents. *DEUS EX* accommodates a fairly continuous space of viable strategies and tactics to overcome obstacles. On the other hand, the options presented to players during narrative cutscenes and dialogs are always presented as discrete choices between a handful of options (see figure 6.9), which have little effect on the development of the narrative events. The progression of the levels is always the same, sometimes dialogs and cutscenes are different, but the overall outcome is usually the same. Only during the very final level, the actions of the player will determine which one of three endings is selected.

The common implementation of dialog trees can further serve as an example

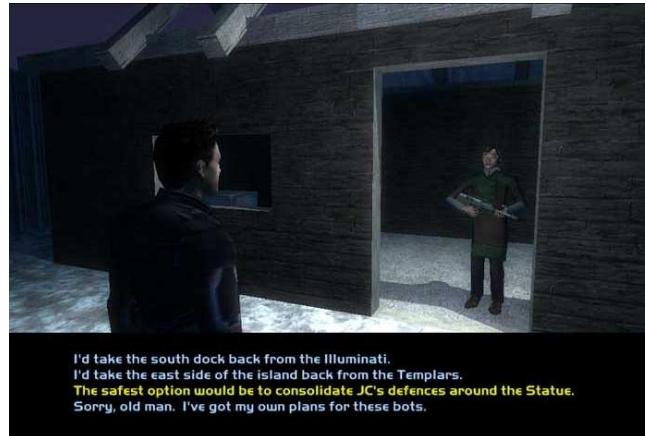


Figure 6.9: A dialog in DEUS EX where the player can choose between four options.

of the problems faced by designers of interactive missions and stories in games. Dialogs feature in many games, and while certain games do not even bother to make their dialog interactive, those that do often resort to using dialog trees. A dialog tree offers players a few optional lines to advance the dialog. Reaching certain leaves in a dialog tree might change the story state. Many dialog trees are not really trees, but are more akin to directed graphs as often different paths through the tree will take players to the same node in the dialog. One problem commonly associated with these tree-like structures is that they quickly become inefficient and overly complex; the number of options that need to be created is much larger than the average player will ever see, and without proper editing tools the writer of a dialog tree might quickly lose track of all the options. Worse, dialog trees do a rather poor job of really creating the illusion of freedom or agency. With only a few options available at a time, chances are that players will feel constrained in their options (Wardrip-Fruin, 2009, 56). In all likelihood players will recognize the tree-like structure, and it is not uncommon for them to traverse the entire tree in order to explore all possible gameplay options, which mostly is a trivial yet tedious task. In short, at the micro-level of the dialog, these tree-like structures often constitute poor gameplay (Dormans, 2006c). Still, at the macro-level of mission or story, they are quite common.

It is probably one reason why many story-driven games do not offer any variation in the way the story unfolds. Games such as HALFLIFE 2, THE LEGEND OF ZELDA: TWILIGHT PRINCESS, or STARCRAFT, all use their levels and challenges as simple bottlenecks that players must pass in order to advance the story. Failure to overcome an obstacle simply means that players will have to try again. Sometimes there is little room for variation as is the case with STARCRAFT II, but that rarely goes beyond the option to choose in which order levels are completed and the occasional choice between two alternative options. In the ten years that lie between the release of DEUS EX and STARCRAFT II there seems to be little changes.

For Noah Wardrip-Fruin the problem ultimately lies in the shape of the underlying processes: the processes that underly both the dialog tree and larger interactive story/mission implementations are rather uninteresting. He suggests that a new approach to game fiction is warranted and that this approach should be fundamentally different from the quest flags and dialog trees that govern most missions in games. (Wardrip-Fruin, 2009, 76). I propose that a closer inspection of the mechanics that control game missions offers plenty opportunities to arrive at a better shape for interactive missions.

6.5 Mechanics to Control Progression

The mechanics that govern progression through a level can be represented as Machinations diagrams. These diagrams can be added to a space graph in order to provide more detail. For example figure 6.10 represents a rough representation of the Forest Temple space graph as was discussed in the previous chapter. In this representation fights and tests of skills are omitted, the focus is on the items that must be collected to finish the level: on the mechanics that control progression. However, where before keys were connected directly to a lock, keys now activate mechanics represented by Machinations pools and resources, which in turn operate locks in the space graph. Extending space graphs with Machinations diagrams allows the level to be represented in more detail.

The diagram might seem a bit overwhelming at first. The nodes that represent places in the level's space all have a black outline, whereas the nodes representing mechanics all have a colored outline. The translation of a lock and key mechanism into a Machinations diagram is done with two pools and one resource representing the key. This lock and key mechanism is isolated in figure 6.11. In this diagram the key element in the space graph triggers the pool below it causing the resource representing the key to be transferred to the pool on the right and there by activating (unlocking) the lock. Similar mechanics are used to represent those locks that are opened with multiple keys, as is the case with the monkeys in the Forest Temple level. In this case multiple pools and resources represent those keys which must be collected on a single pool to unlock new areas (see figure 6.12). In the forest temple level there are more lock and key mechanisms that are omitted from figure 6.10 because they operate only on a local scale. One of these mechanisms involves the bombling creatures that spawn at particular locations and which explode a few seconds after they have been picked up. The creatures can be used to destroy certain walls. This mechanism is different from the other lock and key mechanics presented thus far. However, it can easily be captured with a mixed graph representing both space and mechanics (see figure 6.13). In this diagram, the bombling element activates a source that produces a bombling resource that the player can use to trigger a drain on a destructible wall, another drain is activated once per five seconds representing the time the player has to use the bombling before it blows automatically. This also means that player might actually fail to perform the task to open the lock. In *THE LEGEND OF ZELDA: TWILIGHT PRINCESS* this failure poses no problem, as the bomblings respawn quickly after they have been

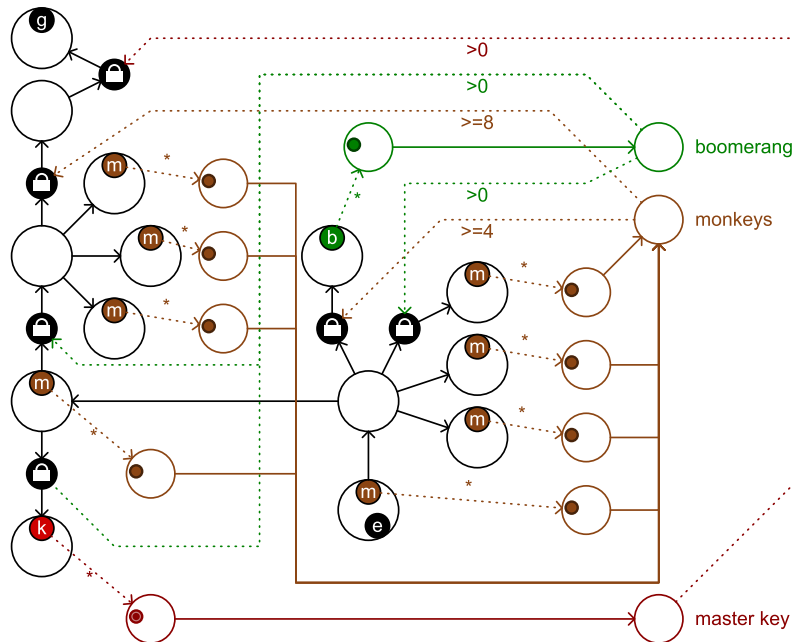


Figure 6.10: A simplified space graph of the Forest Temple level extended with Machinations to represent mechanics. The space nodes and edges have a black outline, the Machinations nodes and edges are all colored. In this figure: b = boomerang, e = entrance, g = goal, k = master key, m = monkey.

used, allowing the player to try again. While the player has a bombling, the source producing new ones is deactivated to make sure there is only one active bombling in play. In addition, the time constraint (bomblings explode a few seconds after the player has picked them up) constrains this mechanism to locks that can be reached within a few seconds after grabbing a bombling. Hence the use of a source and the additional drain in figure 6.13.⁸

In the remainder of this chapter I will focus on the theoretical possibilities and difficulties of relating levels and mechanics using Machinations diagrams to extend mission and space graphs. From the discussion in chapter 4 it became clear that game mechanics benefit from having feedback loops. The lock and key mechanics discussed above do not involve any feedback. Feedback needs a closed circuit that consists of at least one state connection that is not an activator or event; none of the mechanics above fulfill those requirements. There are two design strategies to include more feedback in the game mechanics that control progression through a level: 1) designers could develop lock and key mechanics that involve feedback directly, or 2) progress itself could function more like an abstract resource that can be gained (and might be lost) through mechanics that operate on a fairly large scale. Roughly speaking the first strategy boils down

⁸Although the diagram does not show that players might lose health when they fail to use the bombling in time.

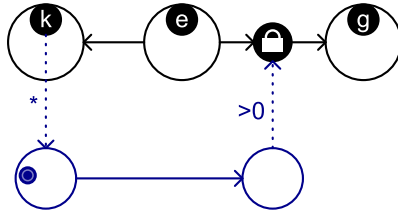


Figure 6.11: The lock and key mechanism in isolation. In this figure: e = entrance, g = goal, k = key.

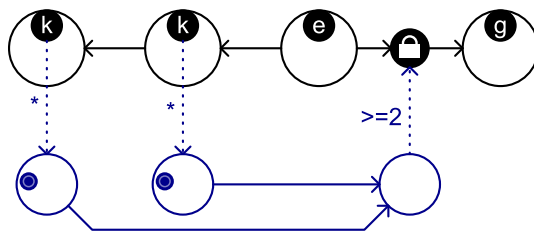


Figure 6.12: The lock and multiple keys mechanism. In this figure: e = entrance, g = goal, k = key.

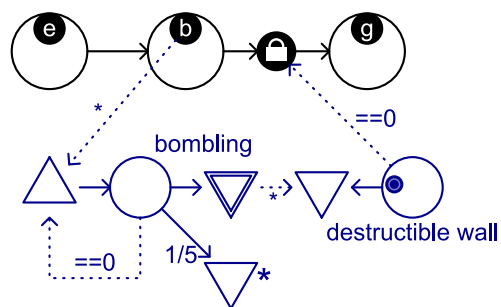


Figure 6.13: The bombing lock and key mechanism. In this figure: b = bombing, e = entrance, g = goal.

to adding interesting feedback mechanics to control locks to a space graph, and the second strategy boils down to replacing missions graphs with more sophisticated mechanics. Both strategies are explored in the following two sections, respectively.

Feedback in the game mechanics that interact with level structures constitute an important and interesting category. As was mentioned in the general discussion of the relation between feedback and emergence in section 1.5, feedback that traverses between different levels of organization within the system contributes to stronger types of emergent behavior. Internal economy and level design constitute such levels of organization. Games that are set up so that their internal economy and level design affect one and other during and as a result of play, have a greater potential for emergent behavior.

6.6 Feedback Mechanisms for Locks and Keys

To create lock and key mechanisms that involve feedback, a good starting point is treating the keys more as a resource that can be produced and consumed. For example, figure 6.14 represents a mechanism where players need to “harvest” ten keys before they can open the lock. Feedback is implemented through the application of dynamic friction on the number of keys players have collected. The more keys that are collected, the quicker the keys are drained. This makes it somewhat harder to estimate how many keys need to be harvested to get past the lock. Obviously, this gets even more difficult as the distance between the location where keys can be harvested and the lock increases. However, the mechanism is not very interesting in itself: it boils down to harvesting enough keys and then make a run for the door, there is little strategy involved.

In an attempt to create a more interesting mechanism, we can apply the dynamic engine pattern (see chapter 4 and appendix B) to a lock and key mechanism. Figure 6.15 represents such a mechanism. This time players need to collect more than 25 keys in order to proceed, but now they have the option to invest 7 keys to increase the harvest rate by 0.5. However, this mechanism is probably too simple, too. It is not very difficult to find out what number of upgrades is ideal for for this scenario.⁹ But even that is not necessary, play-

⁹As it turns out, the ideal number is actually one *or* two upgrades, both arrive at 26 keys

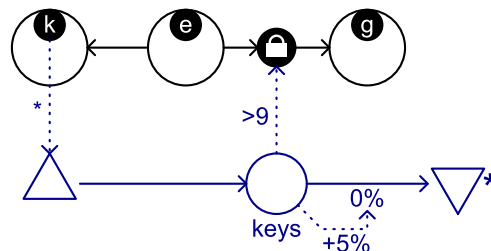


Figure 6.14: Simple feedback applied to a lock and key mechanism. In this figure: e = entrance, g = goal, k = key.

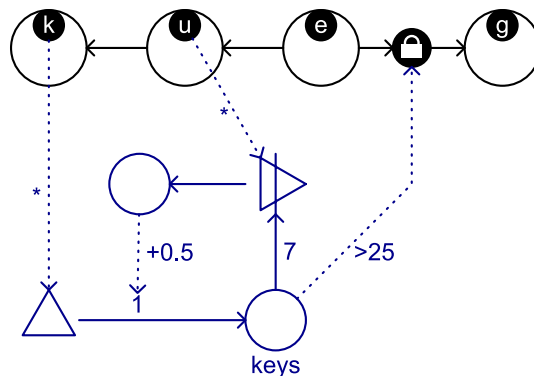


Figure 6.15: The dynamic engine pattern applied to a lock and key mechanism. In this figure: e = entrance, g = goal, k = key, u = upgrade.

ers can achieve the goal without needing to upgrade at all. These weaknesses should not come as a surprise: as I argued in chapter 4 one feedback loop is generally not enough to create an interesting dynamic mechanism. The particular strategies are the direct result of the use of the dynamic engine pattern. Games that do mostly rely on just a dynamic engine as their sole, or single-most important, feedback loop, such as MONOPOLY, usually include random factors to make it more interesting and unpredictable, but that is not the direction I want to explore here.

To create a more interesting lock and key mechanism, we can complement the dynamic engine pattern by some form of dynamic friction (see figure 6.16). In this case enemies spawn that will consume the harvested keys. Now players have to balance between three tasks: harvesting, upgrading and fighting the enemies to keep their numbers down. This is not a trivial task, playing the interactive version of the Machinations diagram¹⁰ is already a fairly interesting challenge. Simply harvesting will probably not bring the player very far, and although it is possible to achieve the goal by switching between harvesting and fighting, this requires players to maintain a delicate rhythm of switching between the two for a long time; it is very hard to accomplish. Players need to find a balance between the three actions in order to reach the goal. When the fighting is made skill-based, then the most effective balance can actually vary between individual players. In its essence the mechanism is very similar to the basic gameplay mechanism of the real-time strategy game I have discussed earlier: players need to balance between harvesting raw materials, fighting, and upgrading.

at exactly the same time, while taking no upgrades or more upgrades turns out to be slower.

¹⁰Also available at http://www.jorisdormans.nl/machinations/wiki/index.php?title=Mission_Mechanics.

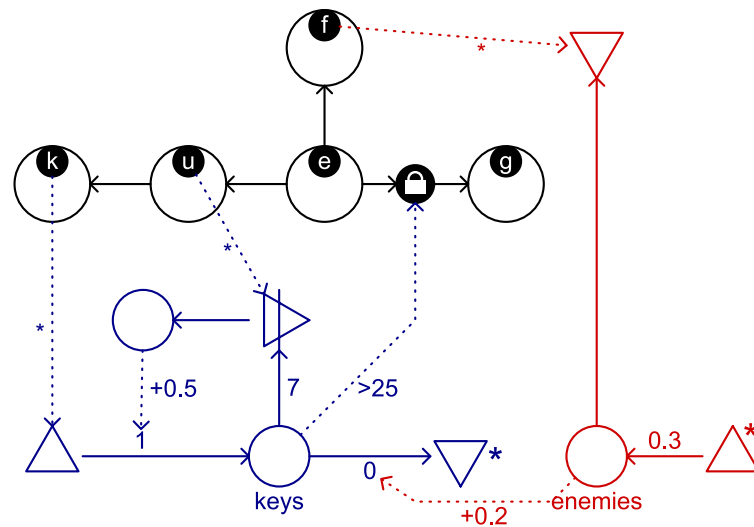


Figure 6.16: The dynamic engine pattern and dynamic friction applied to a lock and key mechanism. In this figure: e = entrance, f = fight enemies, g = goal, k = key, u = upgrade.

6.7 Progress as a Resource

In many games that integrate emergent gameplay with progressive level design, the goal is to reach a certain location. This goal can be represented abstractly with a very simple diagram (see figure 6.17). In its essence, the ‘core mechanics’ of *THE GAME OF GOOSE* are similar. The main elaborations this game implements are the use of dice to determine how much progress the players are making each turn and the chance that a player might gain extra progress, lose turns or lose all progress. More advanced games elaborate more: the most common strategy for action-adventure type games is to make the production of progress non-trivial and interesting in itself. To a certain extent, the experiments with lock and key mechanisms that involve feedback, discussed in the previous section, fall into the same category. This section seeks to go one step further, it explores the possibility to involve progress itself in a mechanism to make the gameplay more dynamic. Put differently, this section explores how feedback mechanics might replace typical missions found in most games of progression.

An interesting and fairly abstract implementation of a progress mechanism can be found in the latest edition of *WARHAMMER FANTASY ROLEPLAY*. The rules of this tabletop role-playing game include the concept of a ‘progress tracker’ as a generic tool to manage the players’ progress towards a single goal, competition for conflicting goals by multiple parties, or even the players’ party’s internal tension and friction. The progress tracker takes the form of a track that can be built from individual track pieces (see figure 6.18). This allows the ‘game master’ to build tracks with lengths that suit the current situation.¹¹ Markers on

¹¹In a tabletop role-playing game one player takes the role of the game master (also known as

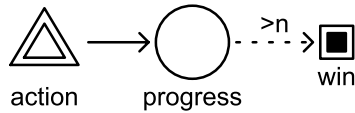


Figure 6.17: A simple progression mechanism

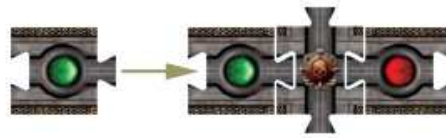


Figure 6.18: Pieces that are used to build a progress track in WARHAMMER FANTASY ROLEPLAY

the track indicate the progress of individual parties. The rules suggest a number of ways a progress tracker can be used to facilitate scenes that involve races, chases, investigations. The tracker can also be used to represent a time limit by forcing the players to complete a certain task before a marker on a progress track reaches the end, or to create tension by using it to track the build-up of some “looming danger” unknown to the players.

Crucially, progress tracks in WARHAMMER FANTASY ROLEPLAY do not only track progress towards some goal (or danger), they might affect gameplay as well. For example, in the scenario that is published with the rules, the progress track to represent the players’ investigation into some secret cult includes a special position. Once the players’ marker reaches this position, the game master should provide the players with an extra hint in order to speed up their progress. This occurrence creates a one-off, positive feedback loop. Similar events on the party tension meter, a progress track that is part of the core rules, can cause the players’ characters to suffer additional stress, fatigue or wounds, causing destructive feedback.

As WARHAMMER FANTASY ROLEPLAY illustrates, progress mechanics can be used to cause feedback, and this is an excellent way to involve progress in the dynamic behavior of the game. However, more sophisticated forms of feedback can be used to evolve this further. A suitable pattern to accomplish this is the escalating complications pattern (see figure 6.19). With this pattern, which is found in simple, emergent games like PAC-MAN or SPACE INVADERS, the player’s goal is to complete a number of tasks. In PAC-MAN this task is to eat all available dots; in SPACE INVADERS it is to destroy all alien invaders. The task is getting progressively harder as the player is making progress. The dots get harder to reach in PAC-MAN while the alien invaders start to move faster and faster as their numbers decrease. A slightly more complex variation of the pattern, called escalating complexity can be found in TETRIS. In this variation some form of complexity is created and it is the player’s task to keep this complexity under control. However as complexity increases this task gets progressively more difficult, usually another mechanism ensures that complexity is produced at an increasing rate (see figure 6.20). In TETRIS the blocks cause

dungeon master or storyteller). Where normal players usually control one character, it is this players responsibility to set the scenes and take care of all other characters in the game. The game master and the players are not opponents; they collaborate in creating an entertaining game experience.

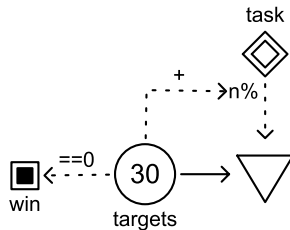


Figure 6.19: Escalating complications: the player's task gets more difficult as there are fewer targets.

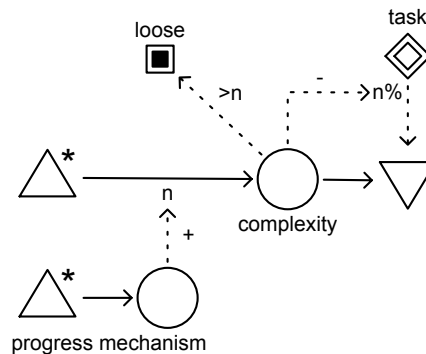


Figure 6.20: Escalating complexity: the player's task gets more difficult as complexity increases.

the the complexity, and the game speeds up every time the player reaches a new level.¹²

Applied to progression, it is possible to model the progress towards a certain goal and have that progress affect the mechanics. In a sense, games of progression have always mimicked this effect by ordering a fixed sequence of challenges roughly from easiest to the most difficult. Nonlinear missions with alternative branches can be built using a similar principle in order to create dynamic levels with more replay-value, but as has been argued before, this strategy is not very effective. Most of the time many more branches and challenges need to be created than an average player is ever going to see. By creating a system where story-like progression emerges directly from the game mechanics, endless possibilities can be created efficiently. When the mechanics are set up to produce enough variety, this could lead to games where interactive experiences, and perhaps stories gain a whole new dimension.

In order to illustrate how a game like this might work, I will discuss an independent Flash game I designed and developed myself.¹³ In this game, SEASONS (see figure 6.21), the player controls a few characters in a 2D platform game constructed as a frame story. In the frame story itself, the player controls a girl that lives in a land that lost all color and is covered in perpetual winter. She travels to her grandfather who tells her stories of the past, when the world still

¹²For a more detailed discussion of escalating complications and escalating complexity see these patterns' description in appendix B.

¹³The reality is that I could not find a published game that could serve as an example. Games where progression truly emerges from mechanics are quite rare, and the examples I know of do not lean towards storytelling. SEASONS has the advantage of being a game that follows the tradition of games of progression, while it does treat progression a little different than most games in this tradition. While developing SEASONS I focused more on the story structure of the game than I explored the emergent properties of the game's ecosystem. Originally it was an experiment with a frame story structure for games first and foremost. At the same time, the contrast between the highly linear stories and the open sandbox-like frame story was deliberate. As was the reflection of the player's progress through the state of the environment in the main quest.



Figure 6.21: Restoring the environment in SEASONS.

had colors and seasons. These stories are also playable levels during which the player controls other characters. By listening to her grandfather's stories the girl learns how the land lost its color and why the cycle of the seasons stopped. The player is encouraged to use that knowledge to slowly restore the land back to life. The girl needs color magic to do so, utilizing color magic, she can restore elements back to their original state. Certain elements, such as trees and living creatures, in return produce new color magic for the player to use. She also needs color magic, to perform double jumps, to fly, or to control the wind, amongst other things, all of which are necessary actions to reach the places where she can restore crucial elements or creatures and to ultimately locate the evil villain. In a way, her progress is reflected in the current state of the environment, the more color is restored the closer the girl is to reaching her goal; in order to reach the villain, the girl needs to restore all colors to the land, and it is through this restoration that the villain is defeated in the end.

When I originally designed SEASONS I was aware of the powerful dynamic engine that is created by its main mechanism: by restoring plants and creatures the player can easily get more color magic. I countered this by adding static friction: restoring plants only works for a limited amount of time. Figure 6.22 represents this mechanism in a Machinations diagram. Apart from the dynamic engine, the most important sources of color magic in SEASONS are reaching certain locations which would unlock and supply the player with a new type of color magic and the stories themselves, as the secret stars the player can find in these stories supply the player with extra color magic in the frame stories.

In hindsight, it would have been more interesting to use a mechanism that would have implemented the escalating complications pattern. For example, by

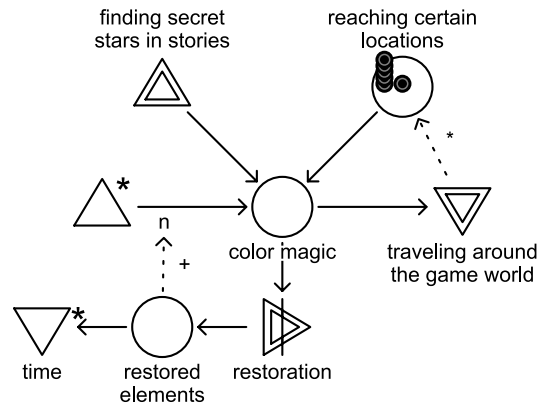


Figure 6.22: The economy of color magic in SEASONS.

having restored elements also feed the aggression of those elements and creatures that have not been restored, requiring the player to spend more color magic in order to avoid them while traveling the world or even to actively oppose them (see figure 6.23). This would in effect aggravate the contrast between those parts of the game world that have been restored and those that have been not restored and cause a gradual increase in difficulty that is an emergent result of the mechanics, and not of careful, progressive level design.

A game set up like this, still relies on triggers to advance the storyline. In SEASONS reaching certain locations in the frame story will unlock new stories to be told by the girl's grandfather and a few other characters in the game. Completing these stories will sometimes unlock a new type of color magic or direct the player to those locations in the game's frame story where they can be unlocked. This set-up works, but much progression is still designed in advance. There are only a few alternative routes the player can take within this structure. This could also be changed. In SEASONS there is a rudimentary ecosystem that produces color magic. This economy could be elaborated so that instead of only one or two restored elements providing the girl with all color magic she needs, the player needs to structure many more elements. By relocating creatures and activating plants the player could build and tune an economy not unlike the city-building in CAESAR III (see above). By making a few changes the ecosystem in SEASONS could be rewired by the player to produce the color magic that is needed for the present task. Reaching certain objectives in this economy could still trigger story events, but the control of these events is far less linear, as it would be fairly easy to design the system in such a way that the player has many options in respect to what objectives to go for first. As with CAESAR III or SIMCITY the game would be able to accommodate a multitude of player strategies, and the game's world would reflect a player's preferences. It would require a multi-part ecosystem that could function in roughly the same way as the basic economy of CAESAR III (see figure 6.6).

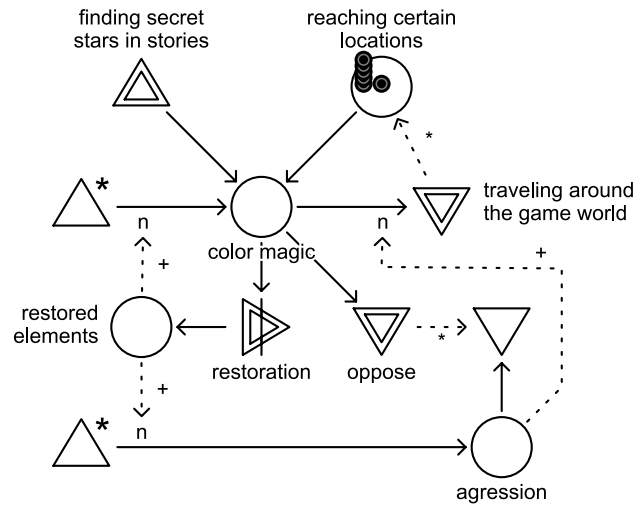


Figure 6.23: A redesign of color magic economy in SEASONS that implements escalation.

6.8 Conclusions

In games, mechanics and levels cannot be separated. Games use level structures to teach the mechanics to players, and use special mechanics to control players' progress through a level. In strategy and management simulation games, players typically have much control over the game economy that is built from the individual mechanics. However, games that find a good balance between the two are rare. One would expect missions to be the place where levels and mechanics converge. But as games have evolved over the past few decades, and ways have been found to create detailed spaces and articulate mechanics, the evolution of missions and mechanics to control progression has mostly stood still. As a result it is rare to find a game that truly integrates emergent gameplay and progressive level design.

In this chapter I have explored the relation between mechanics and levels in order to find a better balance between these two elements of game design. Leveraging the formalisms developed in the previous two chapters, I have explored how mechanics that control level progress could benefit from implementing feedback loops, but also how progress itself can be integrated better in the general economy of a game. These suggestions are preliminary; apart from some promising prototypes they have not been implemented and have not been thoroughly tested. My intention in this chapter was to illustrate how games might integrate these two important elements of game design. The possibilities for the convergence of progression and emergence in games are far greater than the illustrations in this chapter.

The main point, however, is that emergence and progression are not two separate dimensions of game design. Using the right tools designers can shape

emergent mechanics to produce progressive experiences, and by having a clear perspective on a game's internal economy and mechanics designers can structure levels that go beyond a structured learning curve. The internal economy of a game and its level designs constitute different levels of organization. General theory on emergence suggests that feedback that traverses these different levels creates a greater potential dynamic behavior than feedback that operates only within one of these levels. In this way mechanics that drive emergence and progression can be combined to create compelling game experiences that offer great freedom to the player.

Procedural content generation algorithms are hard. Not hard as in you need to get a smarter programmer. Hard as in the travelling salesman ate my NP-complete halting state hard.

Andrew Doull (2008)

7

Generating Games

The last three chapters presented frameworks to deal with game mechanics, and game levels, and discussed their interrelation. Both frameworks are different perspectives on the same object: games. Each perspective foregrounds different aspects of games and their design. Where the Machinations framework foregrounds emergent behavior that stems directly from the structure of the rules, the Mission/Space framework foregrounds progression. Both perspectives are complementary ways of looking at games. When designing a game, designers move back and forth between these perspectives, and in all likelihood also other perspectives such as perspectives that foreground theme, art, interaction, market-value, or suitability for a target audience.¹ These different views on the same game do affect each other. As was already mentioned, what mechanics operate in a game has an effect on how levels might be structured. When designers change the mechanics, this might also lead to changes in the levels and vice versa. This relation suggests that there also might be a relation between the formal models used in this dissertation to represent these perspectives; it suggests formal relationships between Machinations graphs, mission graphs and space graphs that go beyond the possibility of mixing graphs that we saw in the previous chapter. This chapter explores this relationship and discusses how it can be exploited in the creation of powerful, procedural game design tools.

In the first three sections I will discuss model transformation, formal grammars and rewrite systems. This will provide the theoretical background for a formal approach to game design leveraging the multiple perspectives and models. In section 7.4 I will discuss how formal grammars and rewrite systems can be generalized to work with graphs in order to make them applicable to the models that have been discussed in previous chapters. In addition, I will do the same for shapes that are relevant to create actual game geometry in section 7.5. Next, I will illustrate the use of model transformation, grammars and rewrite systems by discussing one important transformation in the process of level design: that of adding locks and keys that allows the designer to transform missions into topological and topographical representations of spaces. In the sections that follow, I will discuss how similar techniques can be used to generate game spaces and game mechanics. In the final three sections of the chapter I will discuss the

¹Needless to say these perspectives do not fall within the scope of this dissertation.

application of these techniques to procedural content generation, which generally refers to the automatic creation of content using clever algorithms. I will discuss three different applications of procedural content generation: the automatic generation of game content, adapting games to player performance and creation of “mixed-initiative” automated design tools. To support this research I have created a number of prototypes that implement the rewrite systems discussed in this chapter. On occasion, I will explain how the theories and ideas have been implemented in these prototypes. These prototypes focused on level generation for action-adventure games first, but their scope widened as the research progressed to include more game content and to support a wider variety of games.

7.1 Game Design as Model Transformation

Model transformation is a notion taken from the practice of ‘model driven engineering’ or ‘model driven architecture’ within computer science. Model driven engineering describes the process of creating software as a series of model transformations where, for example, a business model is transformed into a software architecture, which in turn can be transformed into software code. Model driven engineering is intended to deal with the complexities of designing enterprise-scale software solutions. It depends strongly on a formalized conceptual framework, expressed through different models, which can be used to design systems and communicate about system architecture. It also plays an important role in automatic software generation. One of the main premises is that it relieves programmers from many tedious, manual tasks and elevates the task of programming to a higher level of abstraction, where the most is made of their creativity and ingenuity. Through model driven engineering, the quality and efficiency of software production are to be improved (Brown, 2004). Model driven engineering works with many different models, some of which are specific for a certain domain, while others are more generic. There is a strong push to use Unified Modeling Language (UML) as a standard modeling language independent of platform and implementation (Selic, 2003).

This is not the first time that a model driven approach is explored in the context of game design and development. In a short paper Emanuel Montero Reyno and José Á Carsí Cubel (2008) explore the use of standard UML techniques and tools for the rapid, and mostly automated, creation of game prototypes. They conclude that the UML approach caters better to software engineers than to game designers. In a later paper the same authors sketch a platform-independent modeling language for gameplay specification (Reyno & Carsí Cubel, 2009a). This modeling language still relies heavily on the use of UML, although they do add game structure diagrams and rule set diagrams to the palette of models offered by UML in order to deal with specifics of the domain of games. Using this language, they were able to generate code for game prototypes quickly (Reyno & Carsí Cubel, 2009b). The main difference between their approach and the approach taken here is that they aim to generate games and prototypes of a particular kind (platform games). Their domain is quite

narrowly defined, and there is only little room for design. Their language formalizes a particular subset of games, but not the process of designing them. For the approach presented here, I use model transformations to formalize the design process in a more generic fashion. Automation is only a secondary goal. As a result, the models I am using are not based on UML. They are models specific for the domain of game design rather than for the domain of software production.

The game design process includes many steps during which designers focus on particular aspects of games. Models of some sort have always been an important outcome of these steps. Design documents capturing design vision, early prototypes as a sketch for the intended gameplay, lists of tasks outlining a mission, or hand drawn maps detailing game spaces, all of these are examples of models of (some aspects of) the final game. The steps and products are not separate. Decisions made early on have consequences for later steps. One might say that the labor of a game designer is to take a design vision and to transform it into a working set of game mechanics. Later, these mechanics are transformed into a mission designed to teach players the game. That mission could be transformed into a game space to accommodate it, etcetera. Framed as a series of model transformations, and using clearly defined models that are specific to the domain, this process can be formalized. As it happens, the Machinations framework and the Mission/Space framework provide us with suitable, clearly defined models specific to the domain.

In practice there are many different ways of designing games. The iterations and incrementations towards a finished product are not set in stone. Some designers might start with a premise, design rules to go with it and then proceed to levels and detailed stories. Others might start with a map that constrains the design of game mechanics. Even within the process of designing a game, steps to create particular parts of the game might differ. A tutorial level requires that the game mechanics are clear and finished, but other level content might be dictated by a storyline rather than game options. With applying model transformations to game design, I acknowledge that the process is flexible and different types of games call for different approaches and different types of transformations. However, in order to discuss and illustrate game design as a series of model transformations I have chosen to focus on a particular order of design steps and transformations that in my experience is a sensible way of designing games. In this case, I propose that mechanics are designed before levels and when creating levels, missions are created before spaces. In this process a level designer first creates a mission by generating a list of tasks the player must perform to finish the level, next the designer transforms this mission into a space by arranging these tasks onto a map of the level. The designer then adds detail to the map until it is sufficiently detailed and populated to function as a game level (see figure 7.1). Usually, the later models are more complex and more detailed. One can assume that in this case the original mission is embedded within the representation of space, but not the other way round.

In an alternative approach, not discussed in detail in this dissertation, a designer might begin by designing a space first, and then design a mission that

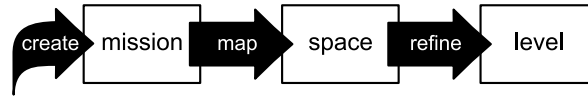


Figure 7.1: Level design as series of model transformations.

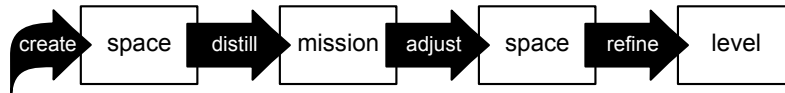


Figure 7.2: An alternative series of transformations.

matches the space, and maybe make some adjustments in order to facilitate that mission before adding detail (see figure 7.2). This approach is better suited to generate levels where spaces conform to some rational or architectural principle outside the mission; a level might be a mine first, furnished with all the elements that one expects from such an environment and then transformed into a space to accommodate interesting gameplay. This way a single space might also host multiple missions, as is the case in *SYSTEM SHOCK 2* where the player traverses the decks of a space ship, and returns to previously explored decks during later stages in the game.

In order for model transformations to work, the models the transformations operate on need to be defined clearly. Model transformation typically uses formal grammars to describe the models. For this reason, I will discuss formal grammars in the next section, and show how they can be used to describe graphs and maps. The transformations themselves are described with rewrite systems, which use a similar type of rewrite rules to describe how a model can be transformed into another model. Rewrite systems are discussed in section 7.3.

7.2 Formal Grammars

Formal grammars originate in linguistics where they are used to describe sets of linguistic phrases that constitute natural languages (Chomsky, 1972). In formal language theory, a language is a, possibly infinite, set of strings. A grammar is a finite characterization of a language. To be more exact, a formal grammar for a language that consists of strings of letters has four elements:

1. A finite set of *terminals* that are elements of the language the grammar is to produce. This set is called the *alphabet*. For example: $\{a, b\}$ is a set consisting of two terminals a and b . It is conventional to use lowercase letters to represent terminals.
2. A finite set of *nonterminals* that are not elements of the strings of the language the grammar is to produce. The grammar's rules will replace these nonterminals with terminals. For example: $\{A, B, S\}$ is a set consisting of three nonterminals A , B and S . It is conventional to use uppercase letters to represent nonterminals.

3. A symbol from the set of nonterminals that is the *start symbol*. It is conventional to use the symbol S as the start symbol.
4. A finite set of *rewrite rules* that take the form of: *left-hand* \rightarrow *right-hand*. Both the left-hand and the right-hand of the rule are strings that consist of terminals and/or nonterminals. Every rule specifies a subsequence of symbols (the left-hand) that can be replaced by another sequence of symbols (the right-hand).

A formal grammar specifies the set of possible strings of terminals that can be generated by a finite number of applications of the rewrite rules starting from the start symbol. For this reason they are also known as generative grammars.

Below is an example grammar that describes a simple ‘mission language’. Note that this example is overly simplistic; it is intended to illustrate the use of formal grammars for strings, it does not make any claim about the structure of missions or the formal grammars that might describe them.

- The alphabet consists of two symbols: $\{g, t\}$, where g stands for goal and t for task.
- The set of nonterminals is $\{S, T\}$.
- S is the start symbol.
- The following rewrite rules apply:

$$S \rightarrow Tg$$

$$T \rightarrow tT$$

$$T \rightarrow t$$

Following this grammar the strings $\{tg, ttg, tttg, ttttg, \dots\}$ are all elements of a set containing all possible missions; they are all part of the language the grammar generates. The grammar describes a mission ‘language’ where a level must have a singular goal preceded by at least one, but possibly more occurrences of tasks. Note that the grammar above is recursive: the second rule creates a string on which the same rule is applicable again. The language generated by this grammar thus contains all strings consisting of a g preceded by at least one, and possibly more t ’s.

The Chomsky hierarchy classifies formal grammars based on the form their rewrite rules (Chomsky, 1959):

- Type 0, or *unrestricted grammars*, have rewrite rules of the form $\alpha \rightarrow \beta$ where α and β may be arbitrary strings containing terminals and/or nonterminals.
- Type 1, or *context-sensitive grammars*, have rewrite rules of the form $\alpha B \gamma \rightarrow \alpha \beta \gamma$ where B is a nonterminal and α, β and γ may be arbitrary strings containing terminals and/or nonterminals. In addition, α and β may be empty strings.

- Type 2, or *context-free grammars*, have rewrite rules of the form $A \rightarrow \alpha$ where A is a nonterminal and α may be an arbitrary string containing terminals and/or nonterminals.
- Type 3, or *regular grammars*, have rewrite rules of the form of $A \rightarrow a$ or $A \rightarrow aB$ where A and B are nonterminals and a is a terminal.

The mission grammar in the example above is a context-free grammar, because all the rewrite rules are of the form that one nonterminal is replaced by a string of terminals and nonterminals. We might specify another mission language, using an unrestricted grammar as follows:

- The alphabet consists of three symbols: $\{g, r, t\}$, where g stands for goal, r for reward and t for task.
- The set of nonterminals is $\{S\}$.
- S is the start symbol.
- The following rules apply:
 $S \rightarrow tg$
 $tg \rightarrow ttg$
 $tt \rightarrow trt$

This grammar generates a language that includes all missions that end with a goal preceded by at least one occurrence of a task, and where rewards can be included if the reward is preceded and followed by a task: $\{tg, ttg, trtg, ttrtrtg, \dots\}$. For generating missions, using an unrestricted grammar has advantages: in this case, generated missions might still be expanded allowing designers, or games, to adjust a mission when needed. When a grammar is context-sensitive, context-free, or regular, a new mission can only be generated from scratch.

7.3 Rewrite Systems

The process through which one model is transformed into another model can be captured using rewrite systems. Rewrite systems share many similarities with formal grammars. Most importantly, rewrite systems also make use of rewrite rules. However, where formal grammars describe and define languages, rewrite systems define particular classes of transformations. These transformations can be used to translate an expression from one language to its equivalent in another, or to elaborate and refine expressions within one language. For example, rewrite systems can be used to transform a mission into a space, or to add detail to existing missions.

In its most generic form, a rewrite system for strings, sometimes called an abstract reduction system or abstract rewrite system, consists of two elements (Klop, 1992):

1. A finite set of elemental symbols A .
2. A finite set of rewrite rules that take the form of: *left-hand* \rightarrow *right-hand* where both the left-hand and the right-hand are strings consisting of symbols from A .

The difference between a formal grammar and a rewrite system is that a rewrite system is not used to generate a language. It has no starting symbol and does not distinguish between terminals and nonterminals. Instead it can operate on any string of symbols; this means that it can operate on an input string specified by a formal grammar. This also means that a transformation described by a rewrite system does not terminate in the same way as the generation process described by a formal grammar does. Where generation terminates when the string it is generating only contains terminals, any application of a rewrite rule in a rewrite system results in a string of symbols that is meaningful and might be further transformed. In contrast, the transformation described by a rewrite system terminates when the string is in *normal form*. A string of symbols is in *normal form* if there is no rewrite rule that allows the string to be rewritten to another string. In addition, a string *has* a normal form when for that string there exists a sequence of rewrite operations that generates a normal form. A rewrite system is called weakly normalizing when all symbols have a normal form, and it is called strongly normalizing, or terminating, when all symbols have a normal form and when infinite sequences of rewrite rules are impossible (Klop, 1992, 5-6).

To illustrate the use of rewrite systems using an overly simplistic example, consider the following rewrite system:

- The set of symbols is: $\{f, g, r, t\}$, where f stands for fight, g stands for goal, r for reward and t for task.
- The following rules apply:

$$tg \rightarrow trg$$

$$tt \rightarrow trt$$

$$t \rightarrow f$$

In this case f would be the normal form of t . The set tg has two normal forms: fg and frg . Likewise, tt also has two normal forms: ff and frf . The symbols f , g and r are normal forms. As there are no sequences of rules that could go on indefinitely, this rewrite system is terminal. When this rewrite system is applied to the mission ttt , the application of a randomly selected rewrite rule generates one of the following results: $\{tttrg, trttg, ttrtg, fttg, tftg, ttfg\}$. The set of missions that are generated after the transformation terminates is: $\{ffffg, frffg, ffrfg, fffrg, frfrfg, frffrg, ffrfrg, frfrfrf\}$.

When all strings and symbols have a single normal form, the rewrite system is called *confluent*, which this rewrite system is not. For a rewrite system that is terminal and confluent every starting set of symbols has a unique terminal set. For certain transformations this is a useful property, for others it is not. In the

case of generating games, many transformations are not confluent, and might not even be terminal. As was argued before, a single mission might map to several game spaces and several game spaces might accommodate several missions, this means that the rewrite systems that allow us to transform missions into space, or vice versa, must not be confluent. On the other hand, a transformation describing how a mission can be translated to an isomorphic space must be confluent.

A rewrite system does not define a language or a model; it cannot be used to describe or analyze a game or a level. It can, however, codify design principles: a rewrite system specifies the operations a designer might perform on a model in order to transform one model to the next. When implemented as an automatic transformation, these rewrite systems are very strict; they allow only the operations that are represented by their rules and nothing more. Real-life designers are more flexible, yet they also obey certain restrictions. If the aim is to create a level that is solvable, no designer would place a crucial key behind a lock opened by that same key, as this would create a deadlock.

The advantage of using a rewrite system is that transformations are defined consistently. If the definition is correct, there is no room for mistakes.² Obviously, this depends on the ‘correctness’ of the rewrite system. It requires considerable effort to design rules that never generate inconsistencies, and to verify that this is indeed the case. But it is possible, and as the set of rules is small in comparison to the set transformations they define it often is worth the trouble. Rewrite systems and the transformations they define are recurrent between projects and games. I imagine that a game company or design community develops and refines sets of transformations over the course of many projects. These sets would then represent the accumulated design lore of that company or design community.

As an additional advantage, rewrite systems allow the design process to be (partly) automated. Automated transformations have two more advantages: 1) they can produce different games or levels quickly, enhancing the output of the level designer. And, 2) this output can be used to validate the correctness of the rewrite systems and the underlying principles by quickly generating games and levels and by verifying them either manually or procedurally.

7.4 Graph Grammars

The models used to describe games in this dissertation are graphs, not strings. In order to use formal grammars and rewrite systems to generate and transform these models, formal grammars and rewrite systems need to be generalized to work with graphs as well. Graph grammars that generate graphs consisting of edges and nodes have been described by Rekers and Schürr (1995). Rewrite systems designed to work with graphs grammars have been discussed by Reiko Heckel (2006). In a graph grammar a structure containing one or several nodes and interconnecting edges can be replaced by a new structure of nodes and edges. Figures 7.3 and 7.4 illustrate this process. After a group of nodes has been se-

²Although what might be considered to be correct will vary between games.

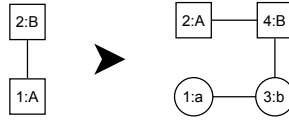


Figure 7.3: A graph grammar rule. Square nodes denote nonterminals and circular nodes denote terminals.

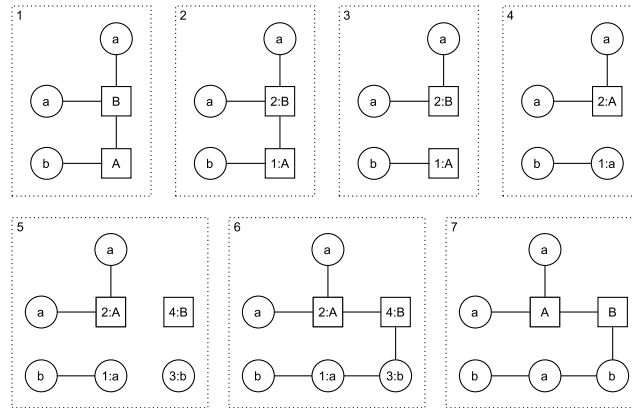


Figure 7.4: The process of applying a rule to a graph.

lected for replacement as described by a particular rule, the selected nodes are numbered according to the left-hand side of the rule (step 2 in figure 7.4). Next, all edges between the selected nodes are removed (step 3). The numbered nodes are then replaced by their equivalents (nodes with the same number) on the right-hand side of the rule (step 4). Then any nodes on the right-hand side that do not have an equivalent on the left-hand side are added to the graph (step 5). Finally, the edges connecting the new nodes are put into the graph as specified by the right-hand side of the rule (step 6) and the numbers are removed (step 7).

Nodes might be deleted as a result of the application of a graph rewrite rule. However, in this case, it is important that the nodes to be deleted are not connected to any other edges other than the ones specified in the left-hand side of the rule.

The graph grammars and rewrite systems used in this dissertation have a form that is analogous to the unrestricted grammars in the Chomsky hierarchy: they have rewrite rules of the form $\alpha \rightarrow \beta$ where α and β may be arbitrary graphs containing terminals and/or nonterminals nodes. This form offers the most flexibility for generating and transforming models representing games in various stages of design.

In order to generate game levels, graph rewrite systems can be used to transform graphs representing missions into graphs representing game spaces. Figure 7.5 depicts mission and space graphs and grammars in relation with each other and a rewrite system. A transformation from mission to space should ter-

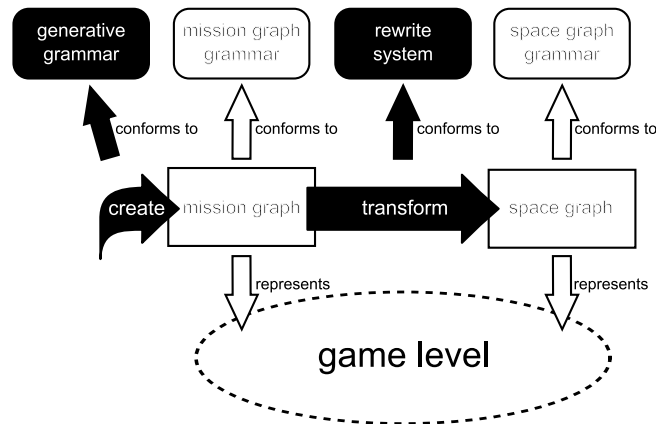


Figure 7.5: Level design as a model transformation.

minate after all mission elements have been replaced with a space element. In other words, in the rewrite system describing this transformation all elements representing mission nodes and edges should all have a normal form that is part of the space graph language. After this transformation the space graph can be further elaborated and refined using different rewrite systems defining the relevant transformations.

7.5 Shape Grammars

Graph grammars can be used to generate Machinations diagrams, mission graphs, and topological models of space. In order to generate game geometry another form of formal grammars, called shape grammars, is required. Shape grammars have been around since the early 1970s when they were first described by George Stiny and James Gips (1972). In shape grammars, shapes are replaced by new shapes following rewrite rules similar to those of formal grammars. Special markers are used to identify starting elements and to help orientate (and sometimes scale) the new shapes.

The implementation of shape grammars in the software prototypes to support this research works with three geometric primitives: points, line-segments and quadrilaterals (quads). In the implementation all shape grammar rules are context-free: the left-hand of any shape grammar consists of one single element, while the right-hand can consist of multiple elements.³ Rewriting works differently for each geometric primitive: the operations for rules that have a point, line-segment or quad as the left-hand element are explained below.

Points only have a location and an orientation. They are represented as

³Creating the shape grammar equivalent of context-sensitive or unrestricted grammars would require a way of detecting the context of a shape in order to identify sets of shapes to be replaced. Where graphs have edges to specify these relations explicitly, shapes lack such an element. For the purpose of generating game geometry context-free shape grammars are sufficient.



Figure 7.6: Points and point rules in a shape grammar.

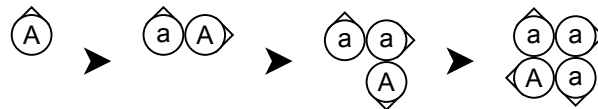


Figure 7.7: A series of transformations based on iteration of rule 2 in figure 7.6.

circles with a triangle indicating their orientation. Colors and letters inside the circle identify the type of point. Capitals indicates nonterminal points, while lowercase indicates terminal points. Shape grammar rules that have a nonterminal point as their left-hand must have at least one point in their right hand, but the right-hand can also consist of multiple points. One point on the right-hand is the starting point. The starting point has a black triangle and replaces the left-hand point and matches its orientation (see figure 7.6). If there are multiple points in the right-hand the extra points are placed relative to the starting point's location and orientation (see figure 7.7).

Line-segments have a location, orientation and a particular length. They are represented as a black line with a triangle indicating their orientation. Colors and letters next to line identify the type. Capitals indicate nonterminal line-segments, while lowercase indicates terminal line-segments. For terminal line-segments the letters and triangles can be omitted (see figure 7.8). Shape grammar rules that have a nonterminal line-segment as their left-hand element must have at least one starting line-segment in their right-hand. The starting line-segment is identified with a black triangle indicating its orientation. In addition, points, other line-segments, and quads can be added to the right-hand. The left-hand line-segment is of a set unit length. To replace a line-segment of arbitrary size and orientation, all right-hand elements are rotated and scaled so that the starting line-segment matches the original line-segment in size and orientation. If the right-hand starting line-segment is gray and dashed it is not placed but only used as a reference to determine the scaling and rotation; in effect, the original line-segment is simply removed (see figure 7.9).

Quads have a location, orientation, and shape. They are represented as a quadrilateral shape with a small square indicating their orientation by marking one side. Colors and letters identify the type. Capitals indicate nonterminal quads, while lowercase indicates terminal quads. For terminal quads the letters

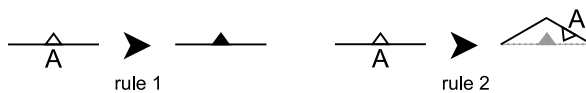


Figure 7.8: Points and point rules in a shape grammar.



Figure 7.9: A series of transformations based on iteration of rule 2 in figure 7.8.

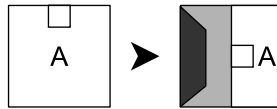


Figure 7.10: A quad based transformation rule.

and squares can be omitted. Quads represent areas in a game, their sides are not barriers. Barriers must be represented with explicitly as line-segments. Shape grammar rules that have a nonterminal quad as their left-hand can have any set of elements as their right-hand. The left-hand nonterminal quad is always shaped as a unit-sized square. In order to replace a quad a complex matrix transformation is used to map the unit-sized square to the shape of the original quad (see Arvo & Novins, 2007). Figure 7.10 gives an example of a rule for quad based replacement. Figure 7.11 depicts a series of transformations based on this rule starting with a different shaped quad.

Combined, these rules can be used to generate quite sophisticated game geometry. Figure 7.12 is an example of a relatively simple grammar. A possible structure that is the result of random application of rules from this grammar is found in figure 7.13. In this case, recursion in the rules creates self-similarity in the structure. In order to prevent such recursion from generating infinitely detailed structures, constraints can be placed on the transformations and replacements. For example, constraints specify how far the right hand is allowed to scale up and/or down in order to match the line-segment or quad to be replaced. Disallowing new line-segments and quads to overlap existing elements in the structure, or be placed outside certain bounds, are other useful constraints to prevent illogical and unwanted results.

There are more possible implementations of shape grammars than the suggestions above. For certain games it will make more sense to use triangles instead of quads as the primitive shape for two-dimensional objects. For other games it will be useful to support square tiles, or to extend these primitives into the third dimension. The choice for quads and the use of only two dimensions in the prototypes for this research was made for convenience in implementation and ease of designing relevant grammars.

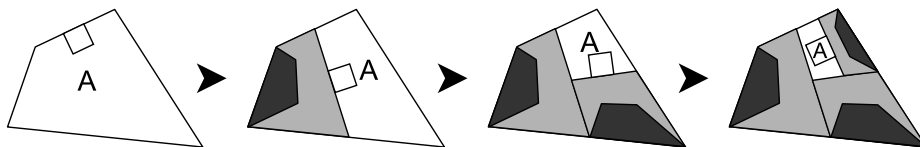


Figure 7.11: A series of transformations based on iteration of the rule in figure 7.10.

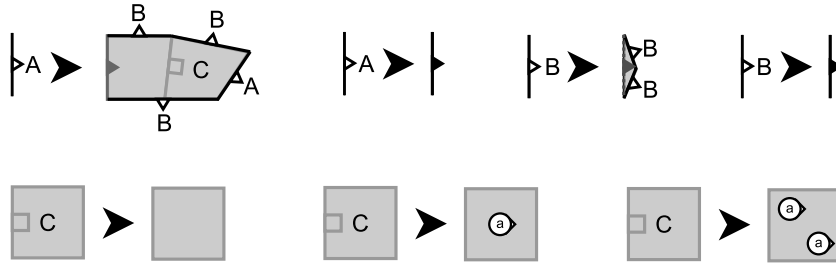


Figure 7.12: Rules for a shape grammar.

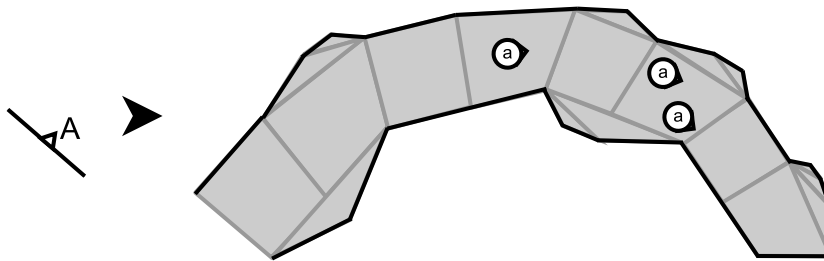


Figure 7.13: An example transformation based on the random application of the rules in figure 7.12.

7.6 Example Transformation: Locks and Keys

Rewrite systems can be used to describe useful, recurrent transformations in game design. This section describes how rewrite systems can be used to add locks and keys to a mission graph and use these locks and keys to transform the mission graph into a non-isomorphic space graph. As was mentioned in section 5.6, this transformation constitutes one of the most important design principles in action-adventure games such as *THE LEGEND OF ZELDA*. Locks and keys control players' movement through the level and foreground their progression. Although the locks and keys in *THE LEGEND OF ZELDA* have many different guises, their basic functionality remains the same. Model transformations from mission to space can be leveraged to explain this design principle in more detail.

Essentially, what locks and keys allow a designer to do is to take a linear series of tasks, which by itself would make for an equally linear level, and transform it into a branching structure (see figure 7.14) which lends itself much better for the creation of nonlinear spaces. This transformation can be captured with only two mission graph rewrite rules (see rules 1 and 2 in figure 7.15).

There are plenty of rules that could be added to this basic set in order to generate more interesting levels. For example, a rule can be designed that moves a lock backwards, towards the goal (see rule 3 in figure 7.15). However, this rule breaks with the level design wisdom that is generally better to have the player encounter the lock before the key. Another rule can be designed that allows tasks that are placed after a lock to be placed in front of a key associated with

that lock (rule 4). This will in effect hide the key, making sure that the player needs to accomplish more tasks before finding it. Other options include using multiple keys on a single lock (rule 5) or creating keys that are used multiple times (see rule 6).

The technique of using rewrite systems is highly controllable. If you consider a lock and key combination to be a single task, then none of these rules changes the number of tasks in the level. This way the size of a level is dictated by the length of the initial mission. In addition, these rules also make sure that a lock will always be followed by another element. This can be verified by inspecting the rules: there is no rule that allows the removal of the last node after a lock, and all additional branches that are created end with a key node that is required to proceed elsewhere. This means that all tasks must be completed in order to finish the level. This restriction explains why in rule 6 the second lock uses a different symbol than the first lock. It means that the second lock cannot be moved by rule 2. Should this lock be allowed to be moved by applying rule 2, the first lock ends up leading to a string of tasks not ending in a goal or a key, or even no tasks at all. This situation is undesired as it might cause the number of tasks the player must perform to complete the mission to be reduced (see figure 7.16).

Figure 7.17 shows a few examples of level structures that were generated with rewrite rules depicted in figure 7.15.

7.7 Generating Space

Once a mission structure is generated that consists of multiple tasks with locks and keys, there are several strategies to build spaces to accommodate the mission. In an earlier paper, I described a method that uses shape grammars to define spatial parts which are used to build up a space not unlike a jig-saw puzzle

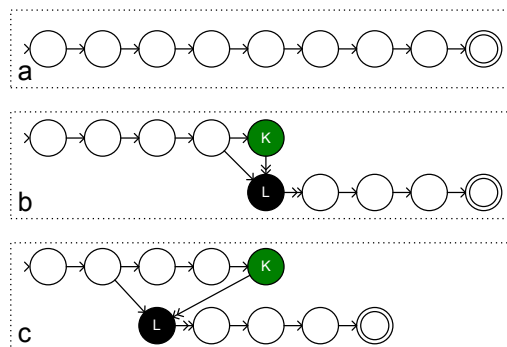


Figure 7.14: Addition of a lock and key transforms a linear mission (a) into a branching structure (b) in which the lock can be moved forward (c).

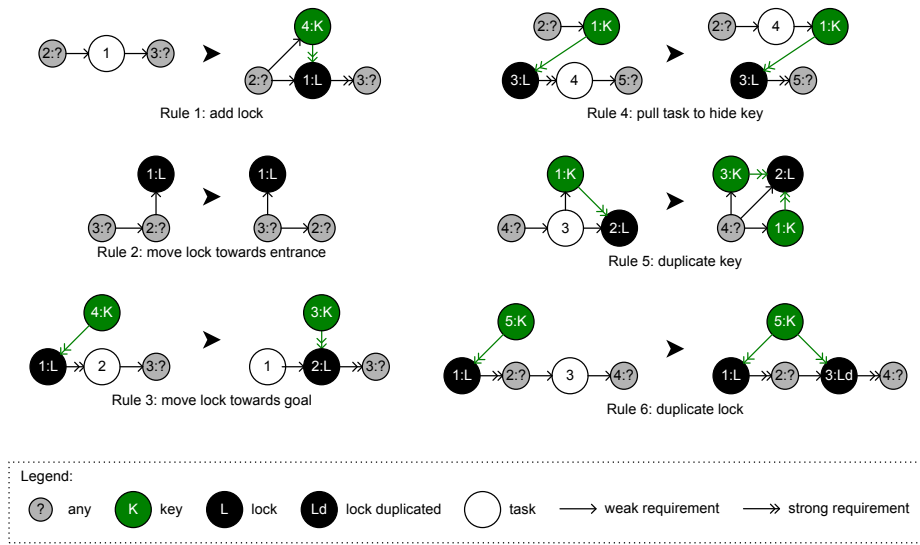


Figure 7.15: Rewrite rules governing the transformations enabled by the use of locks and keys. In these rules, the nodes marked with a question mark can be any node: the question mark acts as a wild card.

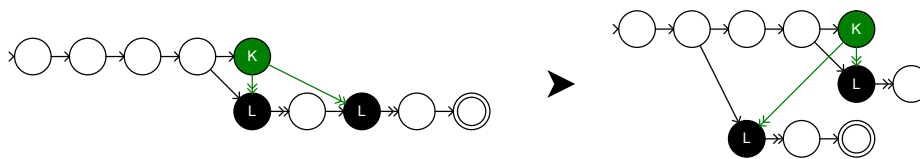


Figure 7.16: Undesired transformation that is the result of applying rule 2 from figure 7.15 to a duplicated lock that is not marked as such.

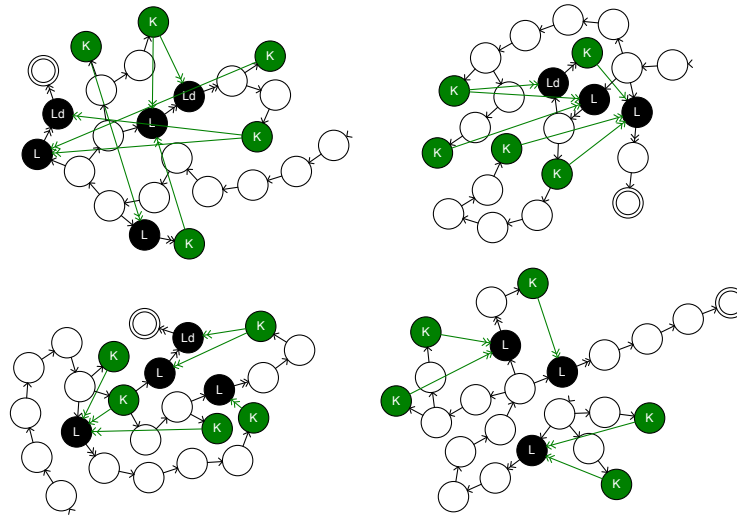


Figure 7.17: A sample of the levels generated by randomly applying the rules 1, 2, 4, 5 and 6 of figure 7.15 on a mission of twenty-one tasks.

(Dormans, 2010).⁴ Although this approach works, it has difficulty generating spaces for missions which allow multiple paths to converge at the same goal. To deal with this problem I take advantage of the spatial nature of a two-dimensional representation of a graph, which can be translated into a shape easily. This approach is also outlined in (Bakkes & Dormans, 2010). The research prototype can generate an organic layout for mission graphs by simulating all nodes in the network as nodes with connections functioning as springs with some basic algorithms to reduce the number of overlapping connections. This algorithm was used to generate the graphs in figure 7.17.

After this, a fairly simple and generic rewrite system is used to replace tasks with places of various sizes, to place keys inside them and, to create locks to connect the places (see figure 7.18).⁵ This rewrite system is both confluent and terminal: it will always generate the same structure for the same input (although in this case the room sizes are set randomly). The result of this transformation is a space graph similar to the one in figure 7.19. From this a spatial structure can be generated that follows the same outline but consists of quads, and line segments (see figure 7.20). This step does not use a rewrite system, as the other steps do. This is because it would require a rewrite system that can mix graph grammars with shape grammars. There is no off-the-shelf solution for rewrite systems to deal with such a mix of two completely different types of grammars. This is complicated further by the fact that all elements of a space

⁴Incidentally, this process is very similar to the “dynamic” level layout of Schell (2008) in figure 5.2 in the previous chapter.

⁵This grammar assumes that tasks that have a direct game element equivalent, such as keys, switches and enemies, are replaced before this grammar is run. For example a task “key” is automatically replaced if there also exists a game element “key”.

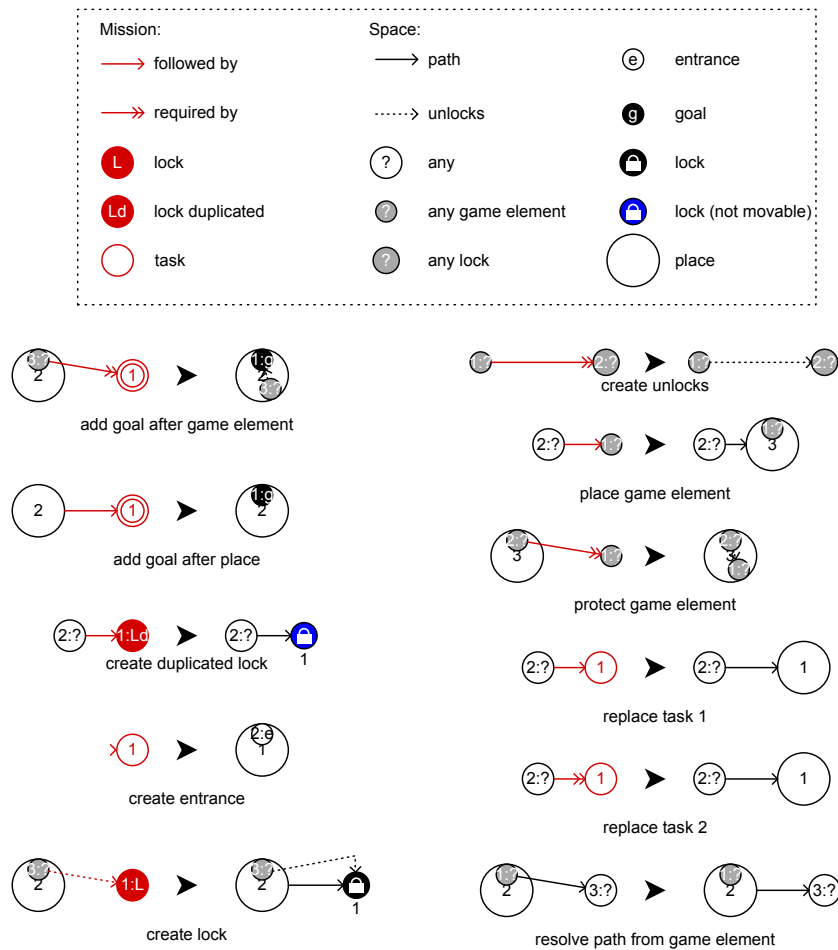


Figure 7.18: A generic rewrite system to transform a mission graph to a space graph.

graph probably need to be transformed into a game geometry simultaneously. Instead, it uses an algorithm designed especially for this step, which is tailored to generate a two dimensional dungeon map. Other implementations might be designed to generate different types of spaces: such as three dimensional terrains, or cities.

The next step involves using a shape rewrite system to flesh out this basic shape and generate more detail. Rules like those in figure 7.12 have been used to add detail to the spatial construction to transform figure 7.20 into figure 7.21. The shape rewrite rules used in this transformation result in a natural looking cave. This need not be the case. With different rules the transformation yields rooms that look much more like artificial constructions (see figure 7.22). In this case the effect would have benefited from aligning the mission structure to a grid before translating it into a spatial construction. After this step, additional

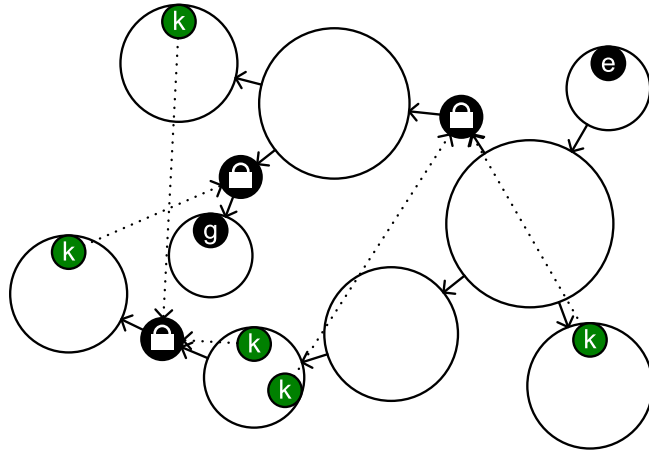


Figure 7.19: Tasks replaced with rooms of various sizes

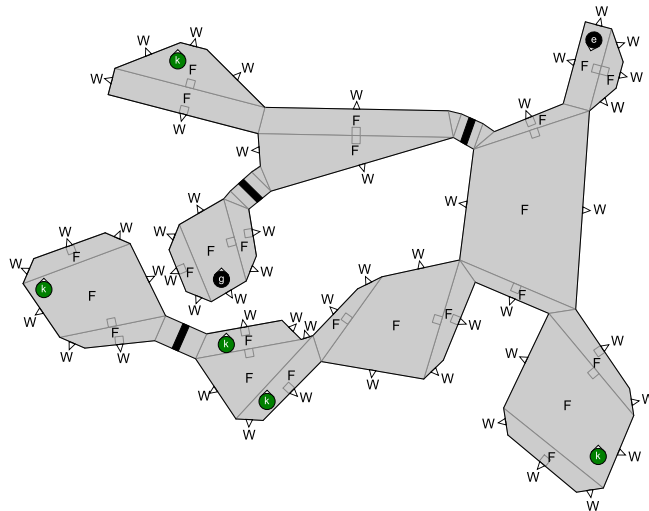


Figure 7.20: The mission structure from figure 7.19 translated into a spatial construction.

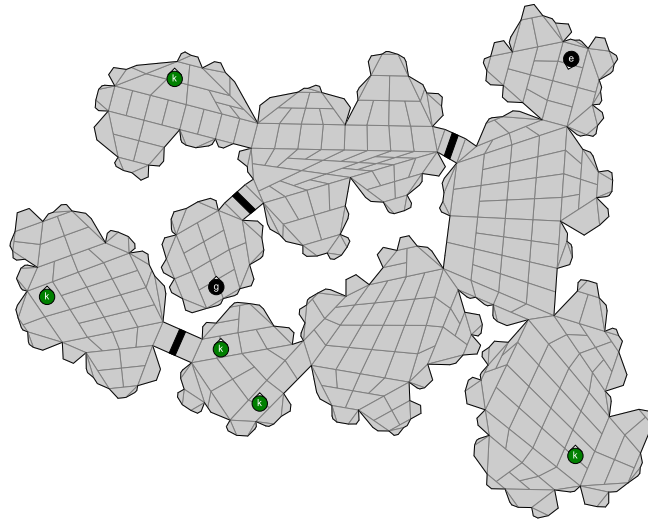


Figure 7.21: Space transformed with shape rewrite system to produce organic dungeon walls.

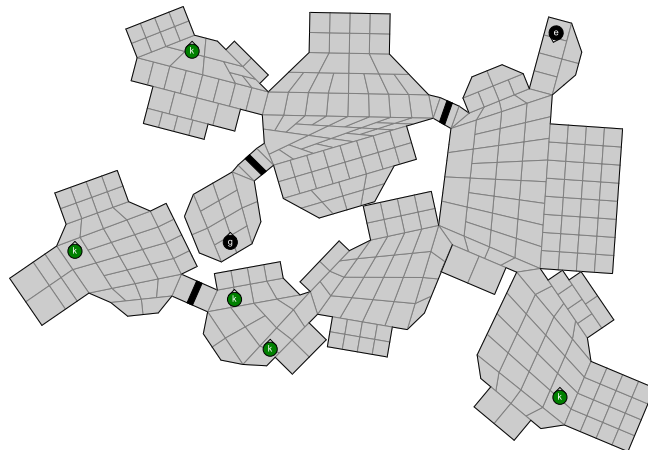


Figure 7.22: Space transformed with shape rewrite system to produce straight dungeon walls.

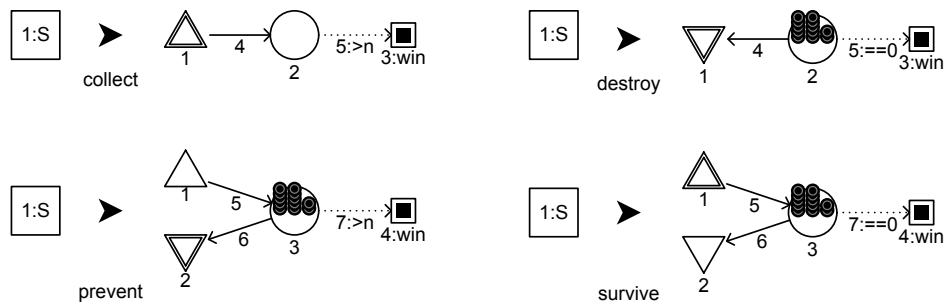


Figure 7.23: Transformation rules to create a goal from an arbitrary starting point (the nonterminal “S”).

transformations can be used to populate the level with treasure, creatures, traps and decorations.⁶

7.8 Generating Mechanics

Just as it is possible to generate missions and spaces using rewrite systems, it is possible to design rewrite systems that generate mechanics from an arbitrary starting point, add mechanics to existing missions and spaces, or generate missions and spaces that match specific game mechanics. Machinations diagrams are an ideal point of departure for such an endeavor as these diagrams are graphs, that can be embedded within missions and space graphs, and can be subjected to the same type of rewrite rules as missions and topographic representations of space.

Rewrite rules can be used to codify recurrent constructions found in games. These constructions include typical game goals, not unlike those described by Björk & Holopainen (2005), Nelson & Mateas (2007), and Djaouti et al. (2008). Figure 7.23 features a number of rewrite rules that might be constructed to include a number of these goals in games. It is not difficult to see that from these starting constructions the mechanics can be expanded by replacing simple mechanics with more sophisticated ones. Examples of rules that describe such transformations can be found in figure 7.24.

Notice that graph rewrite rules for Machinations diagrams require that both nodes and edges have a unique number identifying them for transformation (see section 7.4). Edges in Machinations diagrams behave as nodes in certain respects: they can have textual modifiers and other edges might connect to an edge instead of a node. This means that in many cases simply removing all edges between nodes prior to transformation and adding them after a transformation is not going to work.

It is also possible to include rules that in fact make the mechanics simpler.

⁶In the end, the implementation of shape grammars to generate spaces will turn out to be quite specific to a particular game. It might even be that other types of grammars are used. For example, two-dimensional or three-dimensional tile based grammars will be very applicable to particular games.

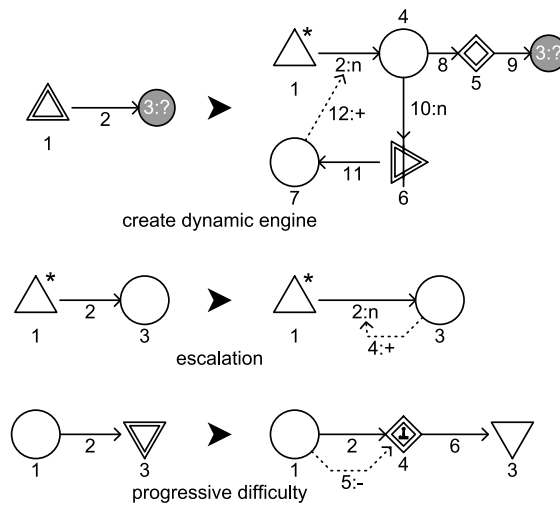


Figure 7.24: Rules to transform mechanics.

As was argued in chapter 2, this can improve the game. Simplifying a game is a stage that almost every design goes through (or should go through), as many elements are usually added during the initial brainstorm. The trick of simplifying, following the discussion in chapter 4, is to cut elements that do not contribute to the structure of the feedback loops present in the mechanics. In this way interesting emergent behavior the game might display is not destroyed during this stage. Rewrite systems are a good tool to guide such a process: for example, they can be used to identify overly complex mechanisms and replace them with simpler ones with equivalent behavior.

Rewrite systems allow mechanics to be generated in close relation with levels and vice versa. For example, it is possible to transform lock and key mechanisms in a space diagram using Machinations and then to use them to elaborate on the mechanics (see figure 7.25). In a similar vein, more elaborate mechanics to deal with other aspects of the game can also be added to mission graphs or space graphs.

As with the transformations used to describe the process of designing a level, there are many different sequences of transformations possible. The most straightforward point of departure is a space graph, and refine it by adding mechanics using rules as illustrated in figure 7.25. However, it might be more interesting to start with mechanics and transform them into a mission that utilizes these mechanics, after which they can be transformed into a space as shown before. In order to transform mechanics into a mission a good starting point would be to associate tasks with all the elements in the Machinations diagram that are interactive (see figure 7.26). Next, these tasks need to be connected in some logical order. This can be done by ‘mirroring’ resource connections with followed by or required by connections in the mission graph, as is suggested by the rewrite rules in figure 7.26. A number of typical Machinations constructions

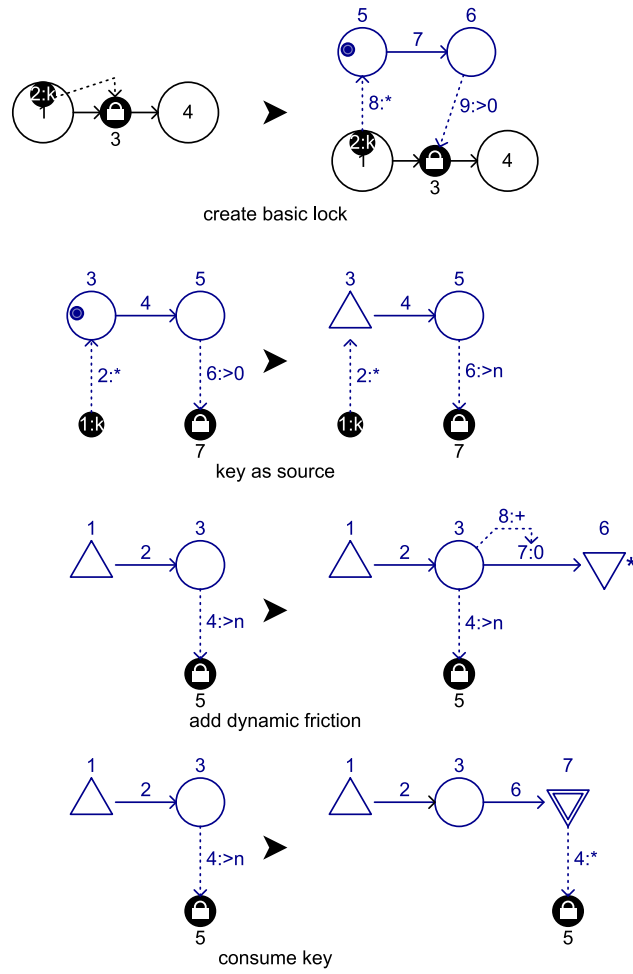


Figure 7.25: Rewrite rules to add and elaborate mechanics for locks and keys in space graphs.

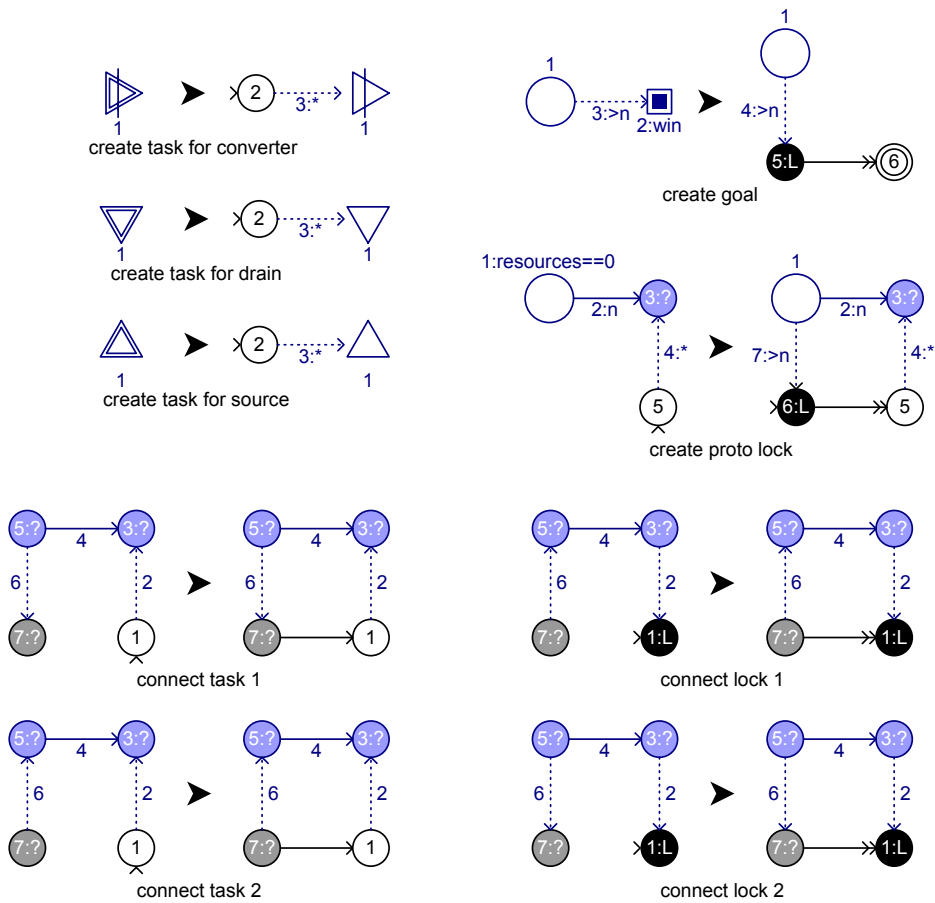


Figure 7.26: Rewrite rules to generate mission structures (black) from Machinations diagrams (blue).

lend themselves for the creation of locks. Missions generated in this way tend to branch pretty wide and might have several starting points. This is only to be expected as missions generated in this way will be more open (or nonlinear) than missions typically found in games of progression.

Figure 7.27 shows the result of a rewrite system as suggested in figure 7.26 applied to a Machinations diagram. The mission generated in this example is relatively simple. However, after the initial generation, missions can be elaborated further using more traditional lock and key mechanisms in order to create levels that mix elements of emergence and progression.

Structuring the learning curve is still important in levels generated in this way. One solution to structure the learning curve can be found in the generation process of the mechanics itself. Assuming this process started out with fairly simple mechanics, like the simple goals presented in figure 7.23, or perhaps with one or two elaborations, a first level, or the first challenges of a level, could

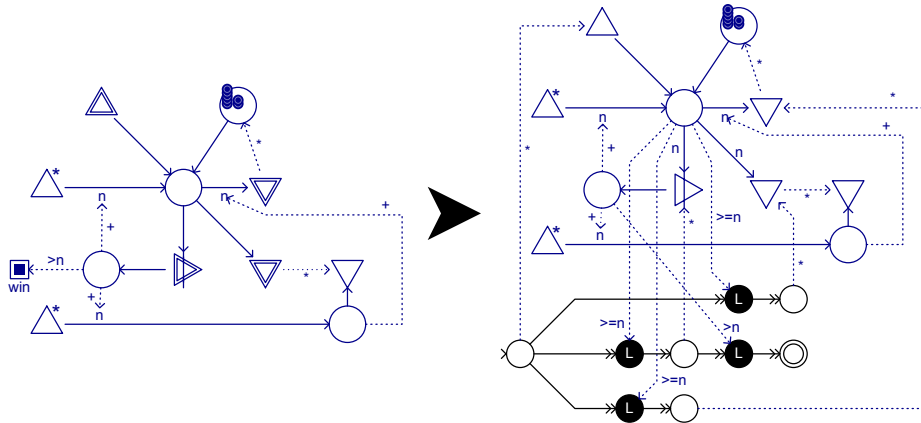


Figure 7.27: Transformation of a mechanics (blue) to mission (black). The mechanics are taken from the SEASONS example from the previous chapter (see figure 6.23).

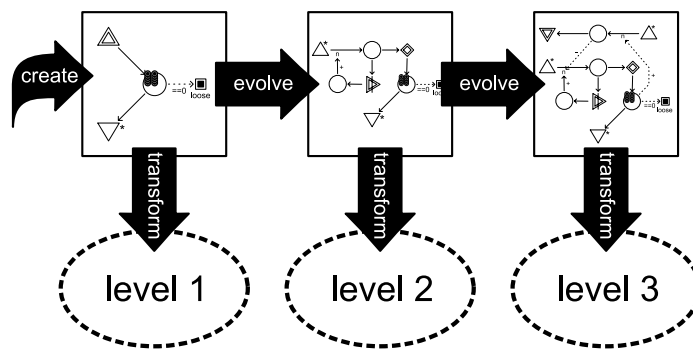


Figure 7.28: Steps in the generation of game mechanics could inform the generation of progressive levels.

be generated from these mechanics. Subsequent levels could be generated from further transformations (see figure 7.28). The transformation history that led up to the complete design of the mechanics could be used as a basis for such a structure of level progression. This way levels might be created that are coherent and where earlier challenges prepare the player for the challenges that are still to come. At later stages of the game, parts of the mechanics might be removed in order to be replaced with new mechanics in order to create variation in the gameplay.

In a way, these transformations describe how games might be designed on a formal level. The framing of game development as model transformation might assist game designers because it helps to structure the way we think about these processes and helps to codify design wisdom in the form of formal grammars and rewrite systems. The additional advantage in describing these processes on a formal level is that this is an important step towards developing tools that

help designers by automating parts of this process; formal grammars and rewrite systems are instrumental in developing tools for procedural content generation.

7.9 Procedural Content in Games

There are multiple ways to look at procedurally generated content in games. The most common application of procedurally generated content is games that generate all or some of their levels automatically with complex algorithms. But these days procedural content generation is also gaining ground as an aid for design teams during the development process or sometimes to assist players to create content while playing.

Games with procedurally generated content have been around for some time. The classic example of this type of game is *ROGUE*, an old *DUNGEONS & DRAGONS* style, ASCII, dungeon-crawling game whose levels are generated every time the player starts a new game.⁷ Newer games that use procedural techniques include *DIABLO*, *TORCHLIGHT*, *SPORE* and *MINECRAFT*. The typical approach of these games can be classified as a brute-force random algorithm that is tailored to the purpose of generating level structures that function for the type of game. Often these algorithms generate a large sample and rely on evaluation functions to select the level that is the most fit (Togelius et al., 2010). Others evaluate the level in order to remove areas that turn out to be unreachable (Johnson et al., 2010). One strategy is to generate a tile map that is filled with tiles representing solid rock and to ‘drill’ tunnels and rooms into the map starting from an entrance. Multiple paths can be created by drilling into new directions from previously created locations. The dungeon is then populated with creatures, traps and treasures.⁸ Another strategy involves zoning the dungeon into large tiles, generate dungeon rooms in some of these zones in the next step, and finally connecting the rooms with a network of corridors.⁹ To create game space to represent wilderness areas, cellular automata can be used to generate more organic structures.¹⁰

Although these algorithms have a proven track-record for the creation of roguelike games, the gameplay their output supports does not necessarily translate to the generation of other types of games.¹¹ For this research the pro-

⁷*ROGUE* is so influential that other games that follow in the same procedural tradition are often referred to as ‘roguelikes’.

⁸See Mike Anderson’s dungeon building algorithm on RogueBasin <http://roguebasin.roguelikedev.com/index.php?title=Dungeon-Building-Algorithm> (last visited July 23, 2011).

⁹See the “Grid Based Dungeon Generator” algorithm on RogueBasin <http://roguebasin.roguelikedev.com/index.php?title=Grid-Based-Dungeon-Generator> (last visited July 23, 2011).

¹⁰See Jim Babcock’s “Cellular Automata Method for Generating Random Cave-Like Levels” on RogueBasin <http://roguebasin.roguelikedev.com/index.php?title=Cellular-Automata-Method-for-Generating-Random-Cave-Like-Levels> (last visited July 23, 2011).

¹¹A typical major component of the gameplay of roguelike games is character building. This type of gameplay, which stems directly from a rather mechanistic interpretation of pen-and-paper role-playing, resolves for a large part around gathering experience points and magical equipment to improve the main character. As game designer Ernest Adams points out in his satirical “letter from a dungeon”, there seems to be little purpose behind these mechanics,

totypes focused on generating content for action-adventure games, which are story-driven games where exploration, puzzle-solving, conceptual and physical challenges make up the majority of the gameplay (Adams & Rollings, 2007). Compared to simulation games and role-playing games, action-adventures typically have a relatively simple set of simulation rules and only a few available power-ups. These games usually do not have an elaborate leveling system where character development, expressed in terms of skills and attributes, is an essential part of the gameplay. Lacking these, action-adventure games must rely more on level design as their prime source of gameplay. As a result, a structured learning curve, clever pacing of action, challenges and puzzles play a more prominent role for the levels in an action-adventure game. A procedure to generate levels for this genre must include a way to incorporate these elements. It is in a similar light that Gillian Smith et al. point out that generating levels for an action platform game requires different techniques, as level design is also a far more critical aspect of that type of game (Smith et al., 2009).

As it turns out, level design principles, like flow, pacing and structured learning curves, are difficult to implement with the algorithms commonly used for roguelike games. These algorithms generally cannot express these principles as these principles mostly operate on larger structures than the individual dungeon rooms and corridors the algorithms work with. The solution of Smith et al. is to create a “rhythm-based approach” to generate levels with “a strong sense of pacing and flow” (2009). The perspective of game design as a series of model transformations provides us with a formal framework that allows us to take yet another approach. Rewrite systems allow us to codify and implement game design knowledge at many different scales. It allows us to start from design, rather than from algorithm, and generate different types of game content (see figure 7.29). What is more, it is applicable beyond automated level generation, which is the focus of most academic efforts. As we have seen in the previous section, it can bridge the gap between levels and mechanics; it is also applicable to procedural generation of game mechanics, which is a relatively unexplored area of procedural content generation (Nelson & Mateas, 2007; Reyno & Carsi Cubel, 2009b). The process is also very flexible, it supports many different sequences of generation. Mechanics might be generated first, or levels might be generated first, or the procedure might switch back and forth between these two. A detailed example of these techniques to generate level structures in the style of *THE LEGEND OF ZELDA* games can be found in appendix C.

Procedural content generation is still a growing research field within game studies and computer science. It is attracting more and more attention as many designers realize that games might benefit from procedurally generated content.

resulting in a shallow representation of character growth as a faint echo of the mythical quest (2000). Gameplay of this type, although forming a viable niche of its own, is well suited for a random dungeon layout. It does not require the same standard of level-design quality as, for example, an action-adventure game from the *Zelda* series. In action-adventure games this style of character development plays only a little part, as is mentioned in an interview by Shigeru Miyamoto, the *Zelda* series main designer (quoted in DeMaria & Wilson, 2004, 240). Just as the random encounter table is an appreciated tool to facilitate a particular style, but not all styles, of role-playing in *DUNGEONS & DRAGONS* (Dormans, 2006b).

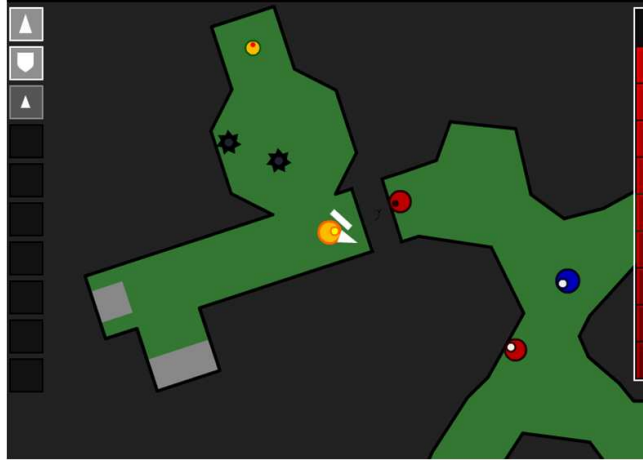


Figure 7.29: One of the prototype games built in support of this research. This game is a 2D action-adventure game with procedurally generated levels, where the player, the light circle equipped with a triangular ‘sword’ and a rectangular ‘shield’, collects items, fights enemies and solves puzzles.

These days triple-A titles require so much content that companies are actively pursuing methods that could lead to shorter development times. The economic benefits aside, there is also an increased interest in the area because procedural content generation could lead to better game experiences. For example Jesper Juul argues that games that generate new levels each time players have to restart again because they failed during the game, reduce the costs of this failure. In general, players dislike having to perform exactly the same actions over and over again (Juul, 2009, 2010). Used in this way, procedural content could lead to games with more varied gameplay.

7.10 Adaptable Games

The generation techniques discussed in this chapter can also be employed to generate content during play, allowing for the opportunity to let the actual performance of the player impact this generation and create games with highly adaptable gameplay. There are several strategies to accomplish this. A straightforward strategy would be to transform certain elements according to rewrite systems as the player plays. The selection of transformation rules could be based on the player’s performance. In this case, the whole level or even the whole game will come to reflect the player’s unique performance.

An interesting example of this technique is discussed by Julian Togelius et. al. (2011) in relation to a SUPER MARIO BROS. clone that features procedural content by the name of INFINITE MARIO BROS. In this game the levels are adapted to players’ actions directly. In one version, whenever the player presses the jump button a platform would be created at that position *in the next level*.

Similarly when the jump button is released the ground is changed, and enemies are added to the next level in response to presses of the fire button. In another version of the same game, the transformations are applied to the same level but just outside the view of the player (or sometimes even in plain view). Extra transformations were added: new enemies are spawned for every coin the player collects and new coins are created for every enemy the player destroys.

In response to the article by Julian Togelius et. al. I created a small experimental version of the classic game BOULDER DASH. In this game, with the uninspired name INFINITE BOULDER DASH, the player collects diamonds from a two dimensional mine that also features boulders and patches of dirt. In contrast to the original game, the game automatically scrolls to the left and the player must keep up with the scrolling to prevent getting killed. The mine wraps around: what disappears on the left reappears on the right. However, as the player collects diamonds new elements are added to the level, including more boulders, diamonds or moving enemies. What element is created depends on the number of diamonds the player collects on each subsequent move, and on the rules of the current level. Figure 7.30 shows the rules for one such level. It indicates that after the player collects the first diamond a boulder is created, after the player collects a second diamond a new diamond is created, two boulders are created after the player collects three diamonds in a row, etcetera. In addition, every time the player collected enough diamonds and leaves the level through the exit, these creation rules change. In general the new level will be more difficult, as more boulders are created, enemies might be spawned, or useful bonuses might become harder to obtain. The player's performance is a factor in this. The game keeps track of the number of diamonds collected, the number of uncollected diamonds, the number of lives lost, among other things and these statistics affect how rules might change. The actual changes are implemented through simple rewrite rules. For example in order to make the game more difficult the rule '3rd diamond = 2 boulders' (collecting three diamonds in a row results in the creation of two boulders) might be replaced with '3rd diamond = 3 boulders', thereby increasing the number of boulders that is likely to be created and thus also the difficulty of the game.¹²

INFINITE BOULDER DASH implements a fairly simple variant of adaptation techniques. The use of rewrite systems to create adaptable games can be taken much further. Imagine a simple vertical space shooter where the behavior of enemies is described by a simple graph (see figure 7.31). Transformations could be used to evolve these graphs to create enemies with different behavior. The game can easily modify this behavior based on the performance of the player: if a player destroys an enemy quickly that enemy is probably very weak, on the other hand if the enemy manages to damage the player it is probably more successful. Particular player actions such as destroying entire enemy waves, the collection of upgrades, or the completion of levels might trigger these transformations. Each trigger might differently affect the probability of transformation rules being selected. At the same time the game could use the same data to trigger similar transformations on the graph describing a level-boss, creating a final adversary

¹²INFINITE BOULDER DASH can be played at www.jorisdormans.nl/InfiniteBoulderDash.



Figure 7.30: In INFINITE BOULDER DASH players collect diamonds in an endless series of generated levels that adapt to the players performance.

that is appropriate for the level. This way the boss would use similar weapons and maneuvers the player already encountered during the level; it would allow the game to prepare the player for the final boss fight.

The game could use a similar technique to build a model of the player. Such a model would start simple, but certain actions of the player would trigger transformations that in turn affect the way the game reacts to the player. This technique could easily be applied in games that feature role-playing elements where the player frequently can choose between options and solutions that represent different ethical attitudes and where non-player characters in the game react to the choices made by the player. Examples of these games are DEUS EX and FABLE. Where most of these games use (relatively) simple variables to track to what extent a certain non-player character likes or trusts the player character (see for example Crawford, 2005), transformations would allow the AI for these non-player characters to build up a far more complex model of the player and act accordingly.

Something similar can be accomplished for game stories, too. When player actions trigger transformations on the general plot, the possible number of generated plots quickly expands beyond the potential of the commonly used branching story trees. Instead of the simple boolean logic that plagues many interactive plots (cf. Wardrip-Fruin, 2009), a game constructed in this way would apply certain transformations on the current plot based on the performance of the player. This could lead to an interaction of much finer granularity between player and game. It could also quite literally lead to an implementation of an interactive structure that Marie-Laure Ryan calls a fractal story where a story keeps offering more and more detail as the player turns her attention to certain parts of the story (Ryan, 2001, 337). Marie-Laure Ryan describes this structure following

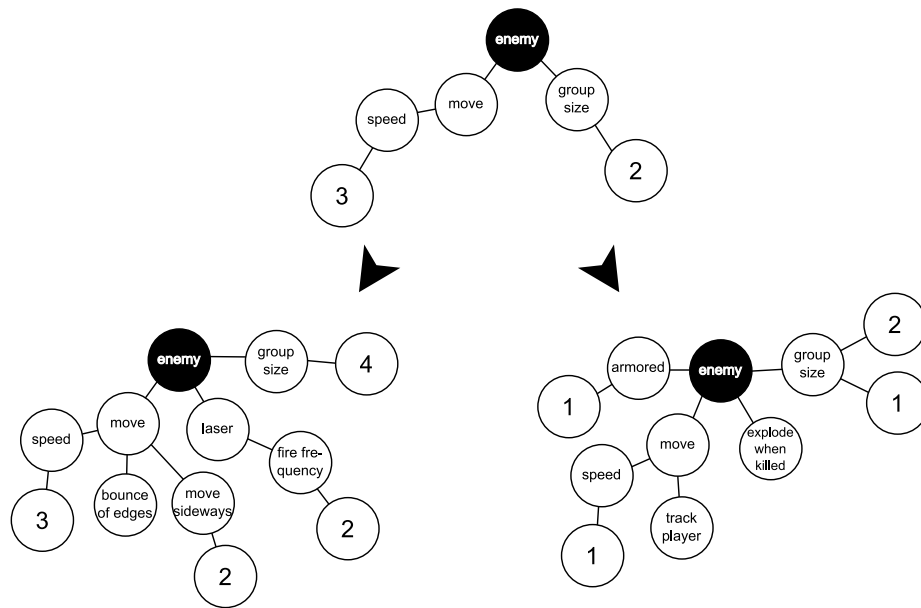


Figure 7.31: Three enemies for a vertical space shooter represented by graphs. The two enemies below might have evolved from the enemy above.

some ideas on interactive storytelling that feature in Neal Stephenson's novel *The Diamond Age* (1995). In this novel a young, lower-class girl called Nell acquires a state-of-the-art interactive storytelling book that teaches her important skills, prepares her for later life and responds to the girl's actual real-life situation. It does so by telling stories of Princess Nell, which take the form of a classic fairytale. The basic premise of the fairytale is always the same, but the details are expanded every time the girl reads further. The story adapts and reflects Nell's life outside the book, helping her to overcome the problems she faces in real life. An important difference between the fractal stories and branching stories is that where branching stories build towards different, pre-designed endings along pre-designed paths, the fractal story transforms itself to accommodate many different paths that essentially lead towards the same goal; the general outline of the fractal story in Stephenson's book is known from the start, when Nell is reading the book the story is not so much advanced as it is expanded (also see Dormans, 2006a).

In order for these techniques to work, the game needs to be able to assess players' actions. Luckily games are typically pretty good at this task; they already reward and penalize players for many actions, for example by rewarding points or taking away lives. The completion of tasks or (sub)quests could easily trigger transformations. Likewise failure to complete tasks or (sub)quests could trigger other transformations. What transformation is selected can be affected further by the model of the game or story as it has been created thus far. Transformation rules automatically become inapplicable when what they need

to replace (the left-hand part) no longer can be found in the current model.

What elements can be replaced with new elements can depend on many things. Rewrite systems can replace any thing in a current representation of a game. However, once a player has encountered certain elements they might need to be excluded from further rewrite operations.¹³ Elements that have not been encountered yet could always be transformed into something else, should the need arise. Even elements that have been encountered, but have not yet been fully explored, might change function or behavior, and if the designer of the game is prepared to risk consistency, anything might be subjected to further transformation.

7.11 Automated Design Tools

Procedural techniques can also be leveraged to automate game design tools. Such tools assist designers to create quality games or game content by automating some tasks of the design process. This approach has been called a “mixed-initiative approach” (Smith et al., 2010; Smelik et al., 2010) and is contrasted with procedural content generation tools that build games and game content without interference of human designers. Although the latter is interesting in itself, there are relatively few games that actually consist of fully generated content. Interest in tools that focus on assisting designers is growing as more and more game companies acknowledge that such tools can increase the effective output of their staff; it allows level designers to focus on the creative aspects of their job and delegate more of the manual tasks to the computer. There are even opportunities for those games that allow players to become the co-creators during play, as is the case with *LITTLE BIG PLANET*.

Model transformations and rewrite systems are an excellent match for the mixed-initiative approach. They provide the designer with many opportunities to control the process of level generation at many different levels of abstraction. At the top most level of abstraction, designers might specify the sequence of transformations, selecting different rewrite systems for each step. In effect this would allow designers to specify whether the level is designed with a particular mission as its starting point (as outlined in figure 7.1) or whether a particular space guides the design of the level (as outlined in figure 7.2). There could even be alternative modules to generate different types of spaces: one rewrite system might generate a ‘dwarf fortress’ while another might generate an ‘orc lair’. Additional transformations might change the ‘dwarf fortress’ into a ‘dwarf fortress overrun by orcs’, etc.

The prototypes that support this research initially all focused on level generation. The rewrite systems that operated in these prototypes were designed to formalize design knowledge such as lock and key structures and learning curves (Dormans, 2010; Bakkes & Dormans, 2010). These prototypes were successful, in the sense that they were able to generate levels quickly and with some interesting gameplay and progression. Where the first prototype had some difficulty in generating levels where different, alternative routes converged, these problems

¹³The version of *INFINITE MARIO BROS.* that does this is very strange, almost unplayable.

were more or less solved in the second prototype. Currently, the implementation of the transformation from a graph that represents a level space to a map is very specific for top down 2D action-adventure style games. The shape rewrite system to refine the space is also implemented in only two dimensions, but, when needed, the same type of grammars and rewrite systems can be made to work in three dimensions.

There are four points that could lead to further improvements:

1. What transformations are involved in the generation process and their order is more or less fixed; in order to make the most of the model transformation approach this should be implemented with more flexibility.
2. Designers could be given more control over the generation approach in order to make the prototype more suitable to a mixed-initiative approach to procedural content generation.
3. Structuring the learning curve can be improved by inclusion of the generation of mechanics as outlined above.
4. The transformation from space graph to a geographical map of the level is a step that is implemented without the use of a rewrite system. Although the step is quite small and rewrite systems control the process before and after, it is something that should be improved for a truly generic application of these techniques. Currently there is no off-the-shelf rewrite system that can deal with topographical graphs and geographic spaces at the same time.

For the third and final prototype (Ludoscope, see section 5.8) I focused on dealing with the first three improvements. Flexibility was created by implementing ‘recipes’ (see figure 7.32). A recipe consists of a series of instructions that can be specified by the user. These instructions include, among others, opening rewrite systems, clearing graphs, applying rewrite rules, and changing the automatic layout settings. A recipe can specify a specific number of times that a rule should be applied, a range from which the tool will randomly select, or it can specify that the rule must be applied as long as there are suitable nodes to apply it to. In this way the user is able to specify the steps involved in the generation process. The interface allows the user to iterate through the steps, to skip certain steps if need be, or to complete the entire process at once. Applying a recipe leads to similar, but different levels.

Manual control over the generation process was also implemented. This feature makes Ludoscope suited for a mixed-initiative approach to content production. Designers can manually select nodes in the graph and then apply any applicable rule to it, or when no node is selected the tool finds out which rules are applicable to any node and offers designers a choice between them (see figure 7.33). When designers choose to apply a rule, a suitable node is selected randomly, unless a specific node was selected. Ludoscope implements an automatic layout system to handle the changing graph representations, but designers can manually change the layout by dragging individual nodes around. In addition, Ludoscope allows designers to directly manipulate individual nodes and

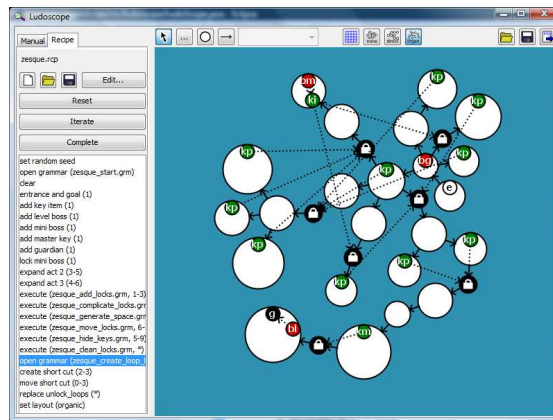


Figure 7.32: Ludoscope, a prototype for an automated level design tool, implements a recipe (on the left) to allow users to specify a multi-step, automatic generation.

edges and well. Designers can add, change or delete nodes as they see fit. To support manual editing, a number of simulation features are implemented that help designers check levels and mechanics for consistency. These features were already discussed in Chapter 5. This way Ludoscope becomes a more general design tool that assists the designer with planning and creating consistent game experiences. What is more, in the current implementation, designers can easily move back and forth between automatic and manual modes of producing game content.

Finally, Ludoscope works with space graphs, mission graphs and Machinations diagrams. With the tool it is possible to create graphs that include elements of all of these different representations of games and use them to represent, and generate, these different aspects of games in unison. It provides designers with a powerful tool, allowing them to experiment and simulate game designs in an early stage of development.

In developing Ludoscope much effort was put into the editors that allow the creation of the various formal grammars and rewrite systems that are involved in the process (see figure 7.34). Creating them is a critical step in automating the design of games. I assume that specific transformations are needed for particular games. Although generic rewrite systems, such as the lock and key grammar discussed above, can serve as useful starting points, I expect that all rewrite systems need to be adapted to a particular implementation. This makes the process of setting up such tools difficult and time-consuming. However, I do believe, that, with some experience, the benefits are far greater than these investments.

7.12 Conclusions

Game design framed as a series of model transformations and the use of rewrite systems allows us to capture and experiment with design principles at

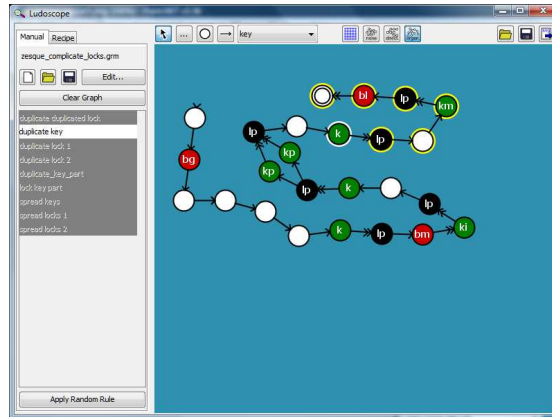


Figure 7.33: Ludoscope. The highlighted rule on the left indicate which rule currently is applicable to selected node in the mission graph (with the white outline).

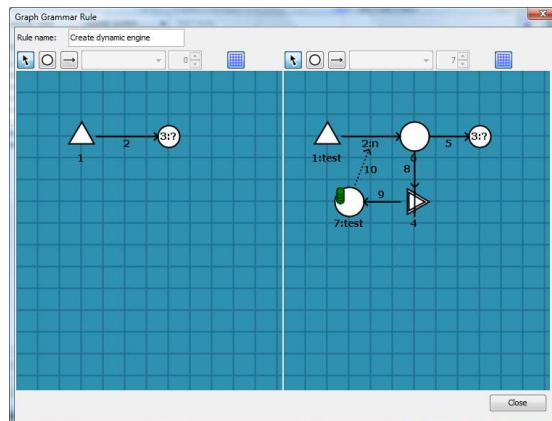


Figure 7.34: Editing graph rewrite rules in Ludoscope.

a formal level. This is applicable in the automatic generation of game content, but also functions as a useful tool in the process of designing levels by hand, or assisting designers by automating parts of the design process. In all cases, it allows level designers to approach their task on a high level of abstraction. At this level of abstraction level designers can focus on the truly creative aspects of their task. This increases their effectiveness in designing levels, and reduces the chance of flaws in the design.

As an example the role of using locks and keys, in various different guises, was discussed in relation to the transformation of linear missions into nonlinear spaces. These transformations can be formalized using rewrite systems, providing us with a structured method to express and experiment with these level design techniques. Adding locks and keys to a mission structure is but one step in the process of designing a level or a game. Many more steps can be defined in a similar way including the generation of game mechanics. The process of designing a game involves many specialized transformations, some of these are only applicable within the context of a single game or genre. By specifying many different steps and individual rewrite systems for each step, a highly flexible body of game transformations can be created. For each individual game, designers might select the transformations that are the most applicable and create a sequence of transformations that yield the best results.

In order to validate the effectiveness of this approach to game development a series of prototypes were created. Initially these prototypes focused on level design and generation, but similar techniques can be used to generate mechanics, adapt gameplay to player performance, and to shed new light on interactive storytelling. What is more, model transformations, formal grammars and rewrite systems suit an approach to content generation where the computer assists human designers by automating certain aspects but leaving the most critical and creative aspects under control of the designers. I expect this approach to gain momentum in the near future as it will help game companies to improve efficiency and quality of their production.

The prototypes outlined in this chapter are still preliminary. There is plenty of room for improvement. As was already pointed out, the transformation from graph to space would benefit from a more generic implementation. Also, to increase the quality of the output, a fitness function, based on game and level design heuristics might be implemented to directly or indirectly affect the transformation process. These things are left to be explored in the (near) future.

Emergence is not the same thing as magic.

Chris Crawford (2005, 138)

8

Conclusions and Validation

This dissertation set out to develop formal tools for the hard task of designing games. Games, as rule based systems, often display emergent behavior; one of the core aspects of games, gameplay, is an emergent property of game rules and mechanics. Many games try to create gameplay with a relative simple set of rules that offer much variation and freedom to players, although not all games do so in equal measure. For games that aim to deliver a structured story-like game experience, level design usually is more critical to the final gameplay than mechanics that build emergent gameplay are. However, these days, games that build on a pure structure of level progression are quite rare; while emergence plays a role in virtually every game.

As rule based systems, games share kinship with simulations. However, where the purpose of simulation is to accurately model (at least certain parts of) a source system, games are often considered stylized representations of systems that might, or might not, exist outside the game. Games can afford many more liberties in order to entertain, to persuade, or to educate. The relation between game rules and the systems they represent, goes beyond accurate, iconic simulation. For games this means that they can use relative simple means to represent dynamic and complex systems. In fact, there are some arguments to suggest that this is where the real power of games lies: games allow us to play with useful shortcuts in understanding complex systems. Yet, as even relative simple games can display complex behavior, games naturally utilize those structural features of rule systems that give rise to the complex behavior in the first place.

Game studies is still a young discipline. Over the years there have been many attempts to create more generic and formal approaches to games and the process to design them. So far, none of these approaches has grown into a standard that spans the game industry and academia. Game design theories face several problems. Perhaps the most prevalent among them is the little trust the game industry puts in the approaches that have been developed over the years. This lack of trust finds its roots in a poor applicability of many of these theories for actual design, especially when one considers that, in general, the investment required to master these theories is quite high. Many theories, frameworks, vocabularies and abstract design tools are more successful as analytical tools

than they are as tools that might help to engineer games. Another problem is that designers can be quite skeptical towards the entire enterprise of developing design methodology; these designers usually dismiss any theory that tries to formalize (and thereby ‘steal the soul from’) the creative process of designing games.

With these concerns in mind I have presented two frameworks and outlined the possibilities to develop automated design tools that are firmly grounded in design practice and aim to assist designers in creating quality games effectively and efficiently. The first framework, *Machinations*, focuses on discrete game mechanics and formalizes a perspective on game rules and emergent behavior in games up until the point where games can be simulated using the *Machinations* tool before they are build. The *Mission/Space* framework focuses on level design. Perhaps it is less innovative than the *Machinations* framework, but it helps to structure the process of designing game levels, nonetheless. What is more, combining both frameworks sheds new light on integrating emergence and progression structures in games that many treat as conflicting ways of creating game challenges. The outline for automated design tools builds on the idea that game design can be framed as a series of model transformations. By using formal grammars to describe the models involved in this process and designing rewrite systems to govern transformations between them, a highly flexible yet formal approach to game design has been created. This approach has been made concrete in a series of prototypes that can generate game content automatically or assist human designers in creating game content.

In this final, concluding chapter I will formulate an answer to this dissertation’s research question, and discuss how all frameworks and prototypes have been evaluated and validated. Finally I will look ahead what research and developments they might inspire in the near future.

8.1 Structural Qualities of Games

The central research question of this dissertation is: *what structural qualities of game rules and game levels can be used in the creation of applied theory and game design tools to assist the design of emergent gameplay?*

The *Machinations* framework has been developed to formulate one part of the answer. For games with discrete mechanics, feedback loops within the internal economy of the game play a critical role in quality of the emerging gameplay. A game’s internal economy is formed by the production and consumption cycles that involve the game’s most important abstract and tangible resources. Depending on the game, these resources might be anything: from weapons, ammunition and health in a shooter game, to food, status and safety in a city building game. Feedback is created when the accumulation, production or consumption of these resources directly or indirectly affect its accumulation, production or consumption in the near future. Feedback can have many different characteristics. Traditionally, positive and negative feedback are distinguished, but for games it is equally relevant whether or not the effect of the feedback is constructive or destructive, at what speed it operates and how durable it is, among other

things.

The Machinations diagrams are designed to represent a game's internal economy and to foreground feedback structures within. From the analyses of a number of games it became apparent that several feedback structures are recurrent and can be found in many games. In addition, most games that display interesting, emergent gameplay feature multiple, but not too many, feedback loops.

To explain the structural qualities of games that are level-driven, internal economy and feedback loops do not help us much. Levels have their own structures that are independent of, yet often also related to, these mechanics. In general level design has not been studied as extensively as game mechanics have been. From a review of a number of level design typologies it has become apparent that one of the problems is that in levels two structures operate that are often only poorly distinguished. To this end, the Mission/Space framework separates missions from space. In this way the framework helps to create a clear perspective on level design. A mission is a series of related tasks players must perform in order to complete a particular level. A space describes the topographic or geographic layout of a level. Missions and spaces are related, but in contrast to what some design strategies suggest, they need not be isomorph. Often missions are laid out in space in such a way, that players are assisted in setting exploration goals, have a fair sense of where they need to go, are rewarded for formulating intentions and plans, and experience the growth of their own, or their characters', abilities.

Lock and key mechanisms play an important role in many games that are level-driven. Lock and key mechanisms create flexibility in the way missions might be mapped to game spaces, breaking away from levels in which mission and space have to be isomorph in order to create a coherent game experience.

Distinguishing between mission and space also helps to identify more clearly the difficulties in designing levels that allow for a more articulate interactive experience. Space and physics have evolved much faster during computer games' relative short history than mechanics to control progression have. By applying the lessons learned from the Machinations framework, more interesting mechanics to control player progression through a mission can be created. Feedback mechanisms that traverse between internal economy and level design can be created in order to push the emergent behavior of games towards areas that hitherto have been relatively unexplored: it helps designing games in which progression is an emergent property of the underlying system.

Although the Machinations and Mission/Space frameworks can be used as analytical tools, they are set up as design tools first and foremost. They help to formalize existing design knowledge and experience, they allow designers to express and discuss designs, but mostly importantly they make tangible certain aspects of game design that normally are quite intangible. Machinations diagrams give shape to a system that otherwise is largely invisible, the Mission/Space framework untangles two superimposed structures that co-exist in level design and creates a clear perspective on both. Both frameworks are fairly easy to learn, they do not rely on extensive vocabularies or pattern catalogs. Yet, despite their simplicity they are quite expressive, and can capture a large

variety of different game structures. I hope that the many examples I have used throughout this dissertation are ample illustration of this.

The concerns about formalizing game design, as expressed by some game designers, cannot be taken away by these frameworks themselves. In a manner, this dissertation does exactly what some of them oppose: it seeks to objectify quality in game design. By pointing out that certain creative aspects within game design cannot be objectified they dismiss any formal approach to their craft. However, the game industry must acknowledge that quality is not equally distributed over all games. We are able to compare games and discriminate between good and bad games. The frameworks presented are designed to help designers identify, create and discuss these qualities in games. These frameworks are not designed to replace creativity, rather, they are designed to *support* it.

8.2 Validation

In this respect, one important question remains unanswered: how successful is the applied theory presented in this dissertation; does it really support game designers in their labor? Clearly, it has not emerged as an industry standard, but nobody can expect that anything could have been developed into an industry standard within the scope of writing this dissertation. The theories have been used by people in the industry and by students, but I have not gathered quantitative data to evaluate its effectiveness. How, then, can the frameworks and the prototypes be validated? How can I be sure that what I have presented is of any real value to a game designer?

I have tried to validate the applied theory of this dissertation in four ways. These ways are described below briefly. In the sections that follow they are discussed in more detail.

1. Implementing many of the theories in software tools to create diagrams, simulate games and generate content, has made these theories much more concrete and has validated them to some extent. On many occasions implementing the theory has led to changes, and vice-versa. In the end I have managed to implement (nearly) all of the theories somehow. The work on visualizing game mechanics eventually has led to the implementation of the Machinations tool, the work on level design has triggered a series of level generation prototypes for action-adventure games that later has grown into a more generic tool for generation of game content: Ludoscope.
2. During the period of this research I have actively sought opportunities to present results to the game development community. In practice this led to a number of talks and workshops. The responses from these workshops were usually positive. Although, I did also meet people who were less enthusiastic for reasons I have discussed in Chapter 3. A number of companies and professional designers have used some of the tools here, and continue to do so.
3. I have presented much of the material as peer-reviewed conference papers

and two journal papers. Many of the chapters in this dissertation are the result of several iterations. For example the Machinations framework started out as a diagrammatic notation based on UML and was initially never intended to be interactive. Many fellow academics, but also people from the industry, have commented on earlier work. Their suggestions led to many improvements and shaped the frameworks into the form presented here.

4. Last, but not least, as I was in the fortunate position to be involved in the development of several game design courses for the Hogeschool van Amsterdam (the Amsterdam University of Applied Sciences) I have had the opportunity to use many of these theories to structure courses and train students.

8.3 Teaching Game Design

Over the past five years I have been involved with the development of several game design courses at the Hogeschool van Amsterdam. In February 2007, my colleagues and I started with a half-year minor program on game design for third year students in the departments of computer science and interactive media. This minor was to complement the already existing game technology minor that, at the time, had been running for two years. This was followed by a serious games course that started in 2008, and a full, four-year game development program starting from September 2009.

My responsibility in these courses have been many, but most of them resolved around setting up and teaching design courses where the students learn how to build rule systems in order create interesting, or meaningful gameplay experiences. Common industry practices have always informed these courses. This includes the MDA framework (mechanics, dynamics and aesthetics), play-centric design, physical prototyping techniques, heuristic evaluations and play-testing strategies (see chapter 3 for a discussion of some of these). But much of the material developed as part of this research also quickly found its way into the course material, starting with the Machinations framework and its precursors.

I have always been a strong advocate of designing games ‘inside-out’. By this I mean that it makes sense to start with designing mechanics, and build the game up from there. The MDA framework suggests something similar. Unfortunately, mechanics and rule systems are not the easiest points of departure for novice designers. It takes experience to appreciate the full effects of small changes to the rules. Lacking experience and an accurate vocabulary to express these nuances, the Machinations diagrams proved to be a useful educational tool. A recurrent assignment was to have students create Machinations diagrams for existing games but also for prototypes they were working on. The first assignment was designed for students to become familiar with the framework, the second assignment to help them get a clear perspective on their own work. These assignments are not easy, for many student it takes time to understand the subtleties of the Machinations diagrams. However, even when students made diagrams that were

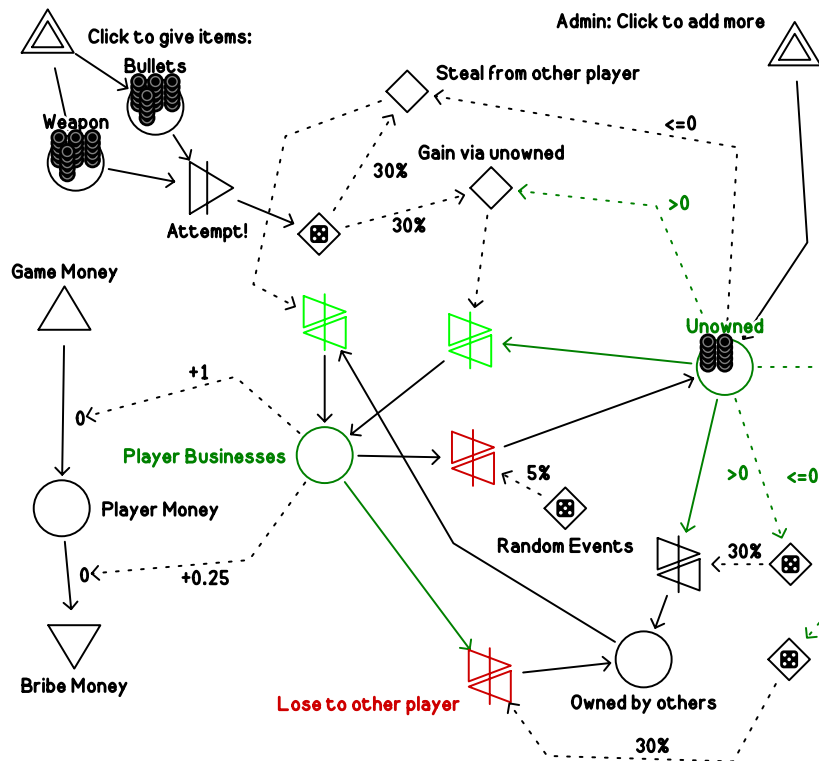


Figure 8.1: A student Machinations diagram by Gerard Meier for the game CrimeBloc (made for Machinations version 2.0)

poor representations of their games, I found that these diagrams were always a good starting point to discuss their designs with great precision and accuracy.

Some students became very skilled in creating these diagrams and kept using them for other assignments as well. These diagrams sometimes turned out to be very detailed and complex (see figures 8.1 and 8.2). But it illustrates that students were able to use them and that at least some of them saw the benefits of keeping to use them. This way, the framework has had an impact beyond the individual workshops and was instrumental in helping students to create a professional perspective on the design of game mechanics.

The workshop that introduces the Machinations framework has evolved into a more or less fixed format. I have used the workshop or variations on it at several occasions. Obviously, it was used during regular classes at the Hogeschool van Amsterdam, but I also used it for workshops I hosted at the Willem de Koning art academy in Rotterdam, at the Game in the City industry event in Amersfoort, workshops at the T-Xchange lab in Enschede, and at Paladin Studios in Leiden (all of these locations are in the Netherlands). During these workshops, participants start with designing a simple board game mechanism that allows them to move pieces on a board, but that also involves some sort of resource.

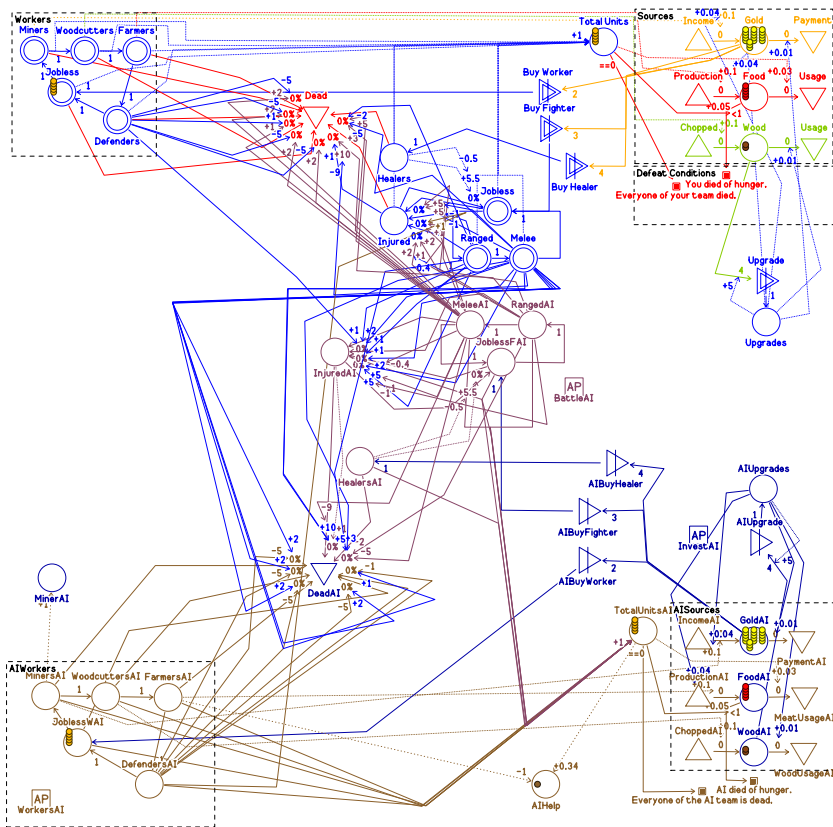


Figure 8.2: A student Machinations diagram by Rik Ruiten. This diagram (made for Machinations version 2.0) represents a fully functional prototype for a real time strategy game.

Next, I explain the basic idea of a game's internal economy and introduce the Machinations diagrams to model this economy. After the participants created diagrams to represent their mechanics in this way, I introduce the notion of feedback and stimulate the participants to explore the concept in their own design. Finally I challenge them to redesign their mechanics in such way that it includes multiple feedback loops. This format is an effective introduction to the framework, and often led to quite interesting designs. Most of the participants were able to create mechanics that were interesting, even when the point of departure was rather simple.

Another, more advanced workshop focuses on the use of recurrent design patterns. This workshop started with a short lecture to explain the framework and then challenged participants to create designs based on a random selection of patterns. Workshops like this were held at a meeting of the Dutch chapter of the Digital Games Research Association in Utrecht in 2008, at the INTETAIN conference in Amsterdam in 2009, and at the DIGRA conference in Hilversum in 2011 (all of these locations are in the Netherlands). This workshop is more advanced and suitable for participants with more experience and expertise.

I was not the only teacher involved in these workshops. My colleagues taught similar classes and workshops. They have similar experiences and point out the advantages of having an online, dynamic implementation visualizing these dynamics. In the near future, we plan to capitalize on this advantage more by expanding our workshop repertoire utilizing this property of Machinations.

The Mission/Space framework was developed later than the Machinations framework. As a result it has not been used as extensively in courses as the Machinations diagrams have been. Yet it has also been instrumental in structuring design courses. The game design course for the second year students in the game development program, was heavily influenced by the ideas developed as part of this framework. The focus of this course was on building prototypes. Every two weeks students needed to finish a new prototype. During each of these two weeks there was a central subject. Mission was one of these subjects, and space was another. During lectures the individual perspectives of missions and spaces were introduced and discussed. For students the identification of lock and key mechanism proved to be a practical and highly applicable perspective that helped structure their designs. Over the years I have received multiple reports that clearly indicate that students are able to work with these concepts, both in analyses and in designs (see figure 8.3).

Framing the game design process as a series of model transformations and procedural content generation has not been the subjects of courses. These subjects are quite advanced, probably too advanced for most of the bachelor students I have worked with. At the moment of writing, the most advanced students are at the end of their second year, and the courses these students will take in their final two years are still under development. Despite the fact that some of these students have expressed a direct interest in content generation, it is unlikely that much of this material will make it in to the program. At best, it might be the subject of some of these students bachelor theses, or play a role in the research program of the game lab that has been set up at the Hogeschool van Amsterdam.

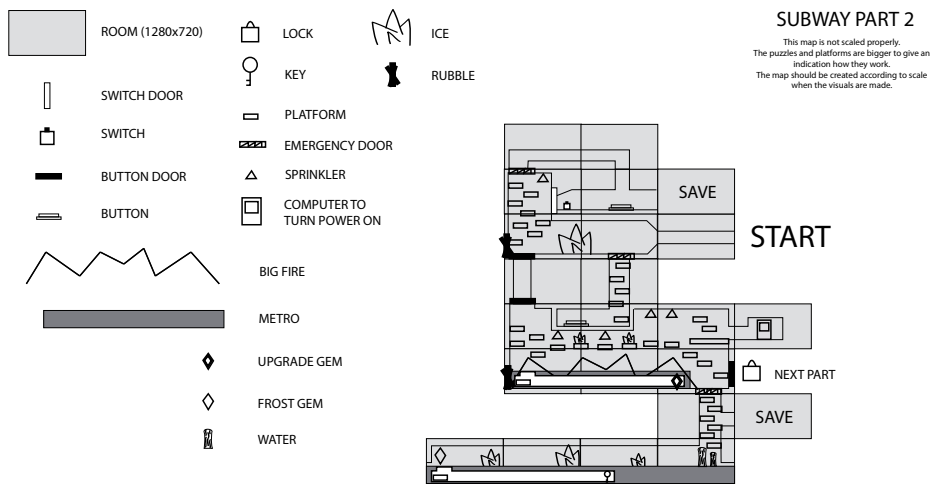


Figure 8.3: One of many level designs by Robin Brouwer and Dennis Kempe for the student game ENCHANTED HANDS focusing on structure and progress. Many of the upgrades and game elements in this level act as locks and keys.

8.4 Building Prototypes

The working examples of the prototypes (many of which are available online) should validate the theory to some extent. Validation always was one of the reasons for the creation of these prototypes. The initial level generation prototypes, in particular, were created with the idea that if a formalized perspective of level design principles would lead to an interesting implementation, then the perspective must be of some value. This remains hard to test, and the success of the levels generated by these prototypes were never subjected to an objective measure. However, general reception of demonstrations of several prototypes has always been quite positive.

Initially, I did not intend Machinations diagrams to be interactive, but I am very happy that I did take that step early. This has led to many improvements in the theory, and it also has led to the creation of a very useful design tool. The fact that it can be used to simulate games is very important in this respect. Representations of games in the Machinations tools can, to a certain extent, be *played*. The fact that in this form the representation retains many of the same dynamic behaviors, is a strong indication that the structures these diagrams foreground indeed play a critical role in creating emergent gameplay.

The development of Ludoscope also proved to be an important test bed for the design theory presented in this dissertation. The theory and the tool evolved in parallel. There were many iterations for both and with each iteration one kept improving the development of the other. This process went on into the very last stages of writing this dissertation, and possibly beyond. Over time, the Mission/Space framework grew more formal, and every time it did, the implementation of the framework made the implementation of Ludoscope leaner

and more powerful. At the same time, the implementation of the framework as an automated design tool kept opening up new avenues for exploration. For example, with mission graphs implemented in Ludoscope it became very easy to add features to check for mission and space inconsistencies. This process is not entirely finished. Ludoscope is still under development. Without doubt, this development will lead to new theoretical insights that will in turn strengthen the development process.

In the end, developing theory and developing software almost became one continuous process. I feel very fortunate that I was in the position to combine these two methods of developing theoretical and applied tools. One might assume that writing and coding was a tough combination but the truth is that the synergy between the two gave me more energy to develop both to a much higher level than I would have been able to do when working on each project individually.

The research also led to the development of a few experimental games. Most of which are never released as they were only rough prototypes. Some of these evolved further, such as the INFINITE BOULDER DASH example discussed in Chapter 7. In other cases, the game projects I have been involved in served as an excellent testbed for the theory and tools presented here. For example, for GET H2O I created Machinations diagrams to help design the game's economy. Games such as SEASONS benefited from my level design research. These examples are discussed in this dissertation. Other projects include ASCENT, a small platform game that was an experiment with abstract resources; DUNGEON RUN, FLIX and FLIX 2 were (at least partly) experiments in level design; CAMPAGNE, a satirical card game that benefited from applying non-iconic reduction and creating a symbolic simulation of Dutch politics; and BEWBEEES that benefited from the Machinations framework in order to create emergent gameplay.

Another experimental game has been built by three students at the Hogeschool van Amsterdam under my direction. For this game, LKE (figure 8.4), I asked the students to experiment with a lock and key mechanism that utilizes a consumable resource as the key and implements multiple feedback mechanisms. In this game the player collects 'key energy' that is consumed in different measures by different doors. In addition, the more energy players have, the more they can see of their environment. At the same time, enemies become more aggressive when the player has much energy. This means that players need to balance their energy level carefully in order to progress through a level safely. From this experiment we learned that this type of construction works well, especially when the 'key energy' also taps into mechanisms, such as the visibility of the environment. The game is still under development and we are still experimenting with more mechanism that consume or are affected by 'key energy'. For this experiment the students built a Machinations diagram in order to grasp the game system, and they found it very useful to balance the system but also to guide further development (see figure 8.5).

8.5 Academic and Industry Reception

The theory presented in this dissertation has grown over the last three years and during that period I reported my progress through the publication and presentation of ten peer-reviewed academic papers. In particular, Chapter 2 is based on a conference paper that I later turned into a journal paper (Dormans, 2008a, 2011a). The Machinations framework and its predecessors were discussed in three conference papers (Dormans, 2008b, 2009, 2011d). The Mission/Space framework evolved while I was working on procedural content and was introduced in a paper on that work (Dormans, 2010). The strategy for integrating structures of emergence and progression outlined in Chapter 6 has been accepted as a conference paper (Dormans, 2011b). The material presented in Chapter 7 was earlier presented in three conference papers and one journal article (Dormans, 2010; Bakkes & Dormans, 2010; Dormans & Bakkes, 2011; Dormans, 2011c).

I also actively sought out people in the game industry and academia to test my ideas through other channels. In some cases these people approached me because they found my work on the Internet or were pointed in my direction by others already familiar with this work. An important outlet for my theories were Machinations design workshops I hosted at conferences and at different companies. These workshops were not very different from workshops I hosted as part of the game development courses at the Hogeschool van Amsterdam, although the pace in these workshops was much higher. The participants of these workshops needed some time to grasp the diagrams, but most of them quickly understood what it was these diagrams try to convey and did see the value of them. Many participants reported that they enjoyed the workshop and that it brought into focus an aspect of game design that normally is not very articulated. Derk de Geus, CEO of Paladin Studios states: “The Machinations framework has been an invaluable tool to visualize our game’s mechanics. Using Machinations’ systems thinking, we have been able to outline complex game systems and identify the strong and weak points in the mechanics. I highly recommend Machinations for designing games and other interactive systems. As an example, I’m seriously considering to use it as a tool for business process modeling” (2011, from personal correspondence).

The person that most strongly influenced the development of the Machinations tool is Stéphane Bura. He contacted me in early 2009 about my first attempts to visualize game mechanics. As Bura himself has an interest in the matter and contributed to the debate himself (2006, also see Chapter 3), he was immediately enthusiastic about the premise of creating visual diagrams to represent internal economies of games. We met shortly afterwards to discuss our work and he immediately encouraged me to develop a tool to express and to execute the diagrams that later would become the Machinations diagrams. After each iteration of the tool and framework Bura was one of the first people to comment and to push me to develop features that would actually help designers to test out designs. This resulted in the features that allow designers to gather data from many simulated runs of which Bura stated: “This is exactly the kind of

tool I need to balance the economies of the games I'm working on" (2010, from personal correspondence).

Experienced developers frequently see the educational value of the framework as its strongest point. As is made clear by Isaac Jeppsen, a system design engineer at Aptima, Inc. who wrote: "I was ecstatic to find your paper on *Machinations*. I've spent the past year at Aptima (a human factors engineering company) trying to explain to them exactly what you illustrated so clearly in your paper. After being a producer at a game development shop, I was continually frustrated by the lack of good tools to explain to my new academic colleagues the essence of a game as the underlying systems that drive it" (2011, from personal correspondence). This would indicate that the framework is more useful for students and novice designers. In a way I agree with this sentiment. Experienced designers need not rely on aids like these frameworks as much as inexperienced designers do, just as programming paradigms and design patterns seem second nature to experienced software developers. Although, I do like to point out that adding the option to simulate games using these framework is a more recent and deliberate attempt to increase the value of the frameworks for more experienced designers.

Finally, during the 2010 G-Ameland game jam I had the opportunity to show and discuss my work with game design veteran Ernest Adams. His books on game design have had a strong influence on the *Machinations* framework, as the notion of internal economy that lies at the heart of the *Machinations* framework was conceptualized by Adams in the first place. Adams immediately saw the potential of the diagrams: "*Machinations* is the best practical design and testing tool for game mechanics that I've ever seen. It's much more convenient and intuitive than using spreadsheets or writing code" (2011, from personal correspondence).

8.6 Omissions

The *Machinations* framework and the *Mission/Space* framework do not cover all aspects of games. There is more to games than just mechanics. Art, artificial intelligence, and interactive control schemes are some of the areas that have not been discussed here, but that are relevant for game design. This dissertation focused on mechanics, but not even all mechanics have been discussed in detail. As indicated in chapter 1, this dissertation has zoomed in on the mechanics that build a game's internal economy and control level progression. Mechanics that deal with physics, maneuvering, and social interaction are beyond the scope of this dissertation. However, it is important to consider these omissions at this point, as all of these type of mechanics can be the cause of interesting and emergent gameplay.

The mechanics that govern a game's physics are of a different nature than those governing a game's economy. Because physics mostly deal with continuous rules and simulation, the discrete representation of rules and mechanics in the *Machinations* framework are a poor match. Likewise the graphs of the *Mission/Space* framework do not match the continuous space made possible by

the physics. To involve a game's physics on the same level of abstraction as the Machinations and Mission/Space framework, a new framework should be found or created that can deal with that sort of rules easily and effectively. What structural qualities such a framework should foreground I cannot tell. I suspect feedback structures will also affect physics. After all, continuous, emergent behavior such as the flocking of birds also involves feedback. A formal framework for game physics is a tantalizing subject in and by itself, that requires considerable research.

Something similar can be said of mechanics that govern maneuvering and social interaction. Both comprise large fields of research, yet I do think that in these areas great strides can be made by trying to identify those structural qualities that lead to emergent gameplay. As it stands now, the theory presented in this dissertation does little to explain the quality of a game like GO that seems to derive its quality from maneuvering almost exclusively. I have tried, several times, to create Machinations diagrams for GO but was never successful. Games that rely a lot on maneuvering probably benefit from an approach based on cellular automata. Most of these games have many similar functioning units that do respond to their immediate surroundings.

There are also some blind spots within the frameworks themselves. For example, failure is something neither really takes into account. However, failure is a very common occurrence in games, even though it is often neglected in many contemporary games (Juul, 2010). The frameworks, and especially mission graphs, can be changed so that failure to perform certain task becomes an explicit option and integral part of the Mission/Space framework. As was briefly mentioned in a footnote, relations that specify how failure, or inhibition of crucial tasks, might be undone could be a great addition to the framework. This way design strategies for games that better accommodate failure might be developed.

8.7 Future Research

So, what is next? The development of the Machinations and Mission/Space work with their associated tools and prototypes is not finished. I see six opportunities for future research and development:

1. The Machinations diagrams might be quite complete, but the research into recurrent design patterns utilizing the framework is not. The patterns described in appendix B are just the beginning. Many more patterns might be found, developed and described. I would advocate that the total number of patterns be kept low. This means that existing patterns in the library should be reexamined from time to time. I already dismissed and merged some patterns.
2. The Machinations tool might be expanded to allow users to embed diagrams in user defined elements to be reused in other diagrams. This would expand the expressive power of the tool extensively. However, there are some risks involved in exploring this direction. One of the strengths of

the current framework is the level of abstraction it enforces. By allowing embedded diagrams, the focus on the larger structure might be lost.¹ On the other hand, it would enhance the capacity of the current diagrams into something closer to a visual programming language specific for the design of games. This might present opportunities for developing prototyping tools and programming environments dedicated to the generation of games.

3. I already mentioned that I have used the frameworks and theory in the development of game prototypes. Unfortunately, none of those games is finished at the moment of writing. But that is because of a lack of time, not a lack of applicability of these frameworks. I know I will continue to work on these projects, but I also hope to inspire others to make use of these techniques to create games with adaptable, dynamic gameplay from which story-like progression emerges naturally.
4. Ludoscope, the automated design tool outlined in chapter 7, remains a prototype. I do think that Ludoscope can be developed into a fully-functional, generic game design tool to assist game designers. This would require much work, and plenty of research, but I do foresee that such a tool can be very beneficial to the effectiveness and enjoyability of game design.
5. As suggested in the previous section, formal frameworks might be extended with the development of frameworks for the mechanics of physics, maneuvering, and social interaction. This would allow us to approach more types of games and tap into more sources to engineer emergent gameplay.
6. More work could be made of validating and refining the results of this research. This can be done by further exploring how the applied theory presented here can be used to develop more advanced game design courses. The Mission/Space framework has not seen as much action as the Machinations framework. I do not expect that these frameworks have been exploited entirely. In addition, more could be done to present these theories to a wider, industry audience. That way, these frameworks might gather more support and might evolve further to better reflect the industry's needs.

8.8 Final Conclusions

Designing games remains a hard task, but gameplay can be engineered. The applied theory presented in this dissertation can help designers to get a better understanding of those structures in games that contribute to the creation of emergent gameplay. It can assist them in their labor to design games, but it can never relieve them of the responsibility that comes from the creative effort required. I do not wish to prescribe the way designers should use this material.

¹This was pointed out to me by Chris Lewis in a personal conversation.

Since theories, frameworks, and automated design tools are only as effective as their users, the real challenge lies still ahead.

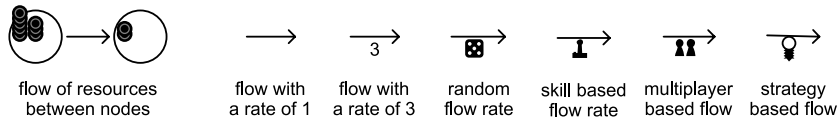
The applied theory on game mechanics and level design aims to elevate game development to a higher level of abstraction using clearly defined, formal perspectives. Assisted by the frameworks and the tools outlined in this dissertation, designers can focus more on those aspects of the process that require their creativity and ingenuity. In the end, I like to think that I have supplied designers with new, more powerful tools, with which their work might become more effective, but also more enjoyable.

A

Machinations Overview

Resource Connections

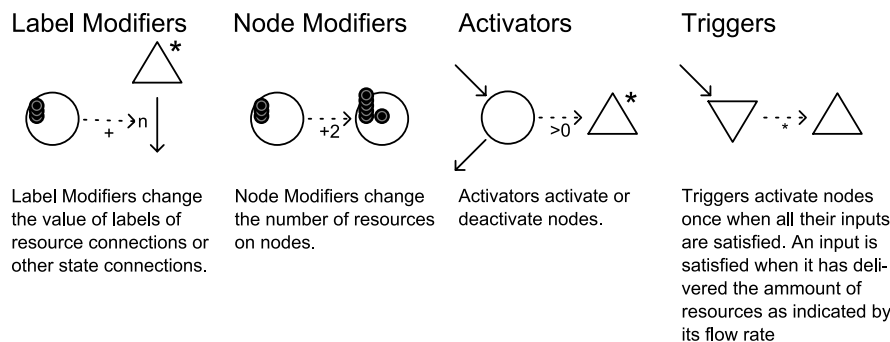
Resource connections dictate how resources flow between nodes.



Label Types	Format	Examples
flow rate:	x	0; 2; 3; 0.5; 1.3
random flow rate:	Dx; yDx; x%	D6; 2D5; D3-D2; 3*D4; 50%
intervals:	x/y	1/4; 2/2; D6/3; D3/(D6+2)
all resources:	all	all
draw randomly:	drawx	draw1; draw2; draw5

State Connections

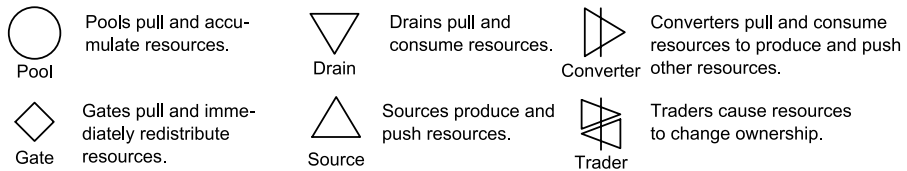
State connections indicate the effects of state and state changes on other elements in the diagram. The state of a node is determined by the number of resources on it.



Label Types	Format	Examples	Applicable to
modifiers:	+; -; +x; -x; +x%	+; -; +2; -0.3; +5%; -2%	value modifiers; node modifiers
interval modifiers:	+xi; -xi	+2i; -1i	value modifiers
probabilities:	x%; x	20%; 3	triggers after a gate
conditions:	==x; !=x; <x; <=x; >x; >=x;	=0; !=2; >=4;	activators; triggers after a gate
range (condition):	x-y	2-5; 4-7	activators; triggers after a gate
trigger marker:	*	*	triggers

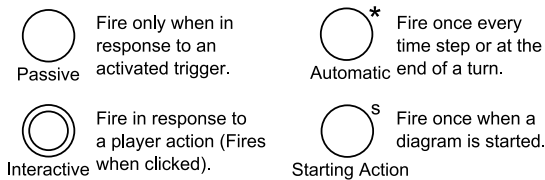
Nodes

Nodes represent game elements that take part in the production, distribution and consumption of resources. Nodes can fire. Firing nodes pull resources according to the flow rates of their input resource connections. A node without inputs will push resources according to the flow rate of its outputs instead.



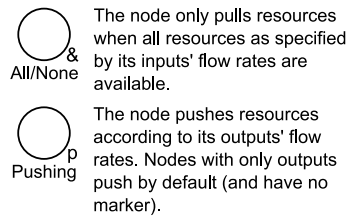
Activation Modes

The activation mode of a node determines when it fires.

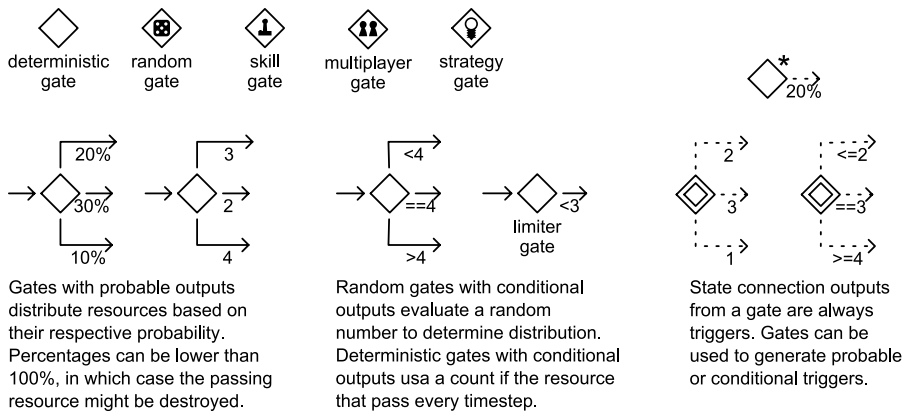


Pull and Push Modes

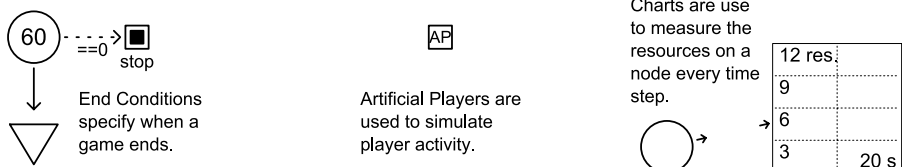
By default, nodes pull as many resources as are available, up to its inputs' flow rates. This behavior can be changed:



Gate Types



Other Elements



B

Machinations Design Patterns

B.1 Static Engine

Type

Engine

Intent

Produce a steady flow of resources over time for players to consume or to collect while playing the game.

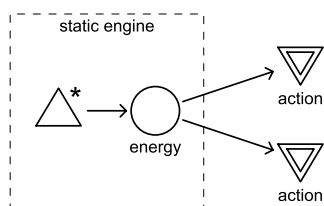
Motivation

A static engine creates a steady flow of resources that never dries up.

Applicability

Use a static engine when you want to limit players' actions without complicating the design. A static engine forces players to think how they are going to spend their resources without much need for long-term planning.

Structure



A static engine must provide players with some options to spend the resources on. A static engine with only one option to spend the resources on is of little use.

Participants

- **Energy** - the resource produced by the static engine.
- **Actions** - a number of game actions the player can spend energy on.

Consequences

The production rate of a static engine does not change. Therefore the effects of the engine on game balance are very predictable. A static engine can only be the cause of imbalance when the production rate of a static engine is not the same for all the players.

A static engine generally does not inspire long term strategies: collecting resources from a static engine, if possible at all, is going to be quite obvious.

Implementation

Normally it is very simple to implement a static engine: a simple source that produces the energy will suffice. It is possible to add multiple steps in the energy production, but in general this will add little to the game.

A static engine can be made unpredictable by using some form of randomness in the production. An unpredictable static engine will force the player to prepare for periods of fewer resources and reward players that make plans that can withstand bad luck. The easiest way to create an unpredictable static engine is to use randomness to vary the output of resources or the time between moments of production, but skill or multiplayer dynamics could work as well.

The outcome of random production rates can be, but does not need to be, the same for every player. By using an unpredictable static engine that generates the same resources for all players the luck factor is evened out without affecting the unpredictability. This puts more emphasis on the planning and timing the pattern introduces. An example would be a game where all players secretly decide how many resources all players can get. The lowest number will be the number of resources to enter play for everyone, while the players who offered the lowest can act first. This would automatically set up some feedback from the games current state to this mechanism.

Examples

In many turn based games the limited number of actions a player can perform each turn can be considered a static engine. In this case the focus of the game is the choice of actions and generally players cannot save actions for later turns. The fantasy board game *DESCENT: JOURNEYS IN THE DARK* uses this mechanism. Players can choose between one of four actions for their hero every turn: move twice, attack twice, move once and attack once, or, move or attack and prepare a special action (see figure B.1).

The energy produced by the spacecrafts in the *STAR WARS: X-WING ALLIANCE* is an example of a static engine. The energy can be diverted to boost

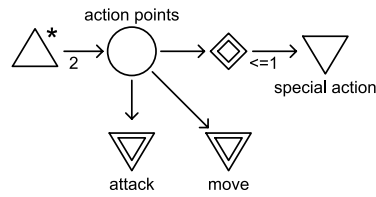


Figure B.1: Distribution of action points in the board game DESCENT: JOURNEYS IN THE DARK.

the player’s shields, speed and lasers. This is a vital strategic decision in the game and the energy allocation can be changed at any moment. The amount of energy generated every second is the same for all spacecraft of the same type (see figure B.2).

FEARSOME FLOORS employs an interesting variation of the static engine. In this game the player controls a team of three or four characters that need to escape the game board. Each turn each character has a limited number of action points which he can use to move and push object around. The number of action points oscillates between two values (that for all character add up to seven). This makes the game surprisingly less predictable, and requires the player to plan ahead at least one turn.

In UP THE RIVER players roll a die each turn and can move one of their ships as many places closer to the harbor. Because the player controls three ships and ships that are not moved eventually fall of the board, the decision of what ship to move is not as straightforward as it might seem at first glance. Elements on the board add extra complications that further emphasize planning and looking ahead.

In BACKGAMMON dice determine how far the player can move up to two of his playing pieces, or up to four if he rolled a double. As the player has fifteen playing pieces in total there are many options. In important strategy in Backgammon is to move the playing pieces in such way that they are safe from the opponent and give the player as many possible options for the next turn.

The GAME OF GOOSE or SNAKES AND LADDERS are poor implementations of an unpredictable static engine as both games lack multiple actions: the player

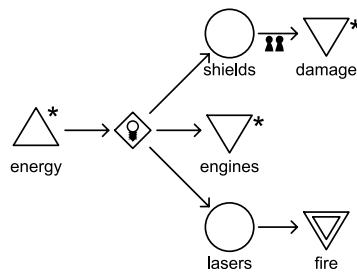


Figure B.2: Distribution of energy in STAR WARS: X-WING ALLIANCE.

does not have any choice and is left to the whims of chance.

Related Patterns

A (weak) static engine can prevent deadlocks in a converter engine.

B.2 Dynamic Engine

Type

Engine

Intent

A source produces an adjustable flow of resources. Players can invest resources to improve the flow.

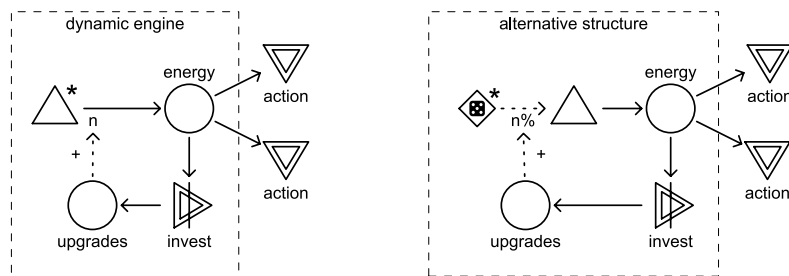
Motivation

A dynamic engine produces a steady flow of resources and opens the possibility for long term investment by allowing the player to spend resources to improve the production. The core of a dynamic engine is a positive constructive feedback loop.

Applicability

Use a dynamic engine when you want to introduce a trade-off between long term investment and short-term gains.

Structure



Participants

- **Energy** - the resource produced by the dynamic engine.
- **Upgrades** - a resource that affects the production rate of energy.
- **Actions** - a number of game actions the player can spend on, including:

- **Invest** - an action that creates upgrades.

Collaborations

The dynamic engine produces energy that is consumed by a number of actions. One action (invest) produces upgrades that improves the energy output of the dynamic engine. A dynamic engine allows two different types of upgrades a player can invest to improve:

- The interval at which energy is produced.
- The number of energy tokens generated each time.

The differences between the two are subtle, with the first producing a more steady flow than the second, which will lead to bursts of energy.

Consequences

A dynamic engine creates a powerful positive constructive feedback loop that probably needs to be balanced by some pattern implementing negative feedback, such as any form of friction, or use escalation to create challenges of increasing difficulties.

When using a dynamic engine one must be careful not to create a dominant strategy either by favoring the long term strategy, or by making the costs for the long term strategy to high.

A dynamic engine generates a distinct gameplay signature. A game that consist of little more than a dynamic engine will cause the players to invest at first, appearing to make little progress. After a certain point, the player will start to make progress and needs to try and do so at the quickest possible pace. The charts these patterns typically generate quite clearly show this behavior. This effect of play is clearly visible in MONOPOLY (see figure B.3, see also section 4.8).

Implementation

The chance of building a dominant strategy that favors either long-term or short-term investment is lessened when some sort of randomness is introduced in the

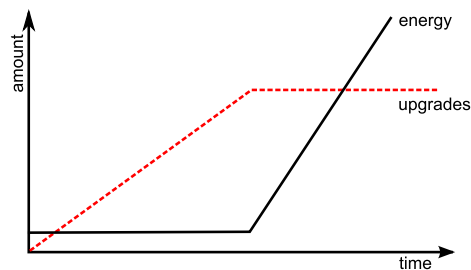


Figure B.3: The gameplay signature of a dynamic engine.

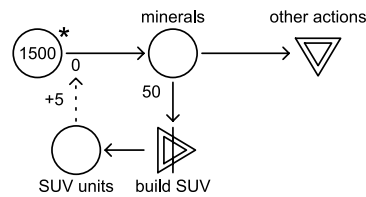


Figure B.4: The harvesting of minerals in STARCRAFT.

dynamic engine. However, the positive feedback loop that exists in an unpredictable dynamic engine will amplify the luck a player has in the beginning of the game, which might result in too much randomness quickly.

The outcome of random production rates can be, but does not need to be, the same for every player. By using an unpredictable dynamic engine that generates the same resources for all players the luck factor is lessened without affecting the unpredictability. This puts more emphasis on the player's chosen strategy.

Some dynamic engines allow the player to convert upgrades back into energy, usually against a lower rate than the original investment. When upgrades are expensive, and the player frequently needs large amounts of energy this becomes a viable option.

Examples

In STARCRAFT one of the abilities of SUV units is to harvest minerals which can be spend on creating more SUV units to increase the mineral harvest (see figure B.4). In its essence this is a dynamic engine that propels the game (although in STARCRAFT the number of minerals is limited, and SUV units can be killed by enemies). It immediately offers the player a long term option (investing in many SUV units) and a short term option (investing in military units to attack enemies quickly or respond to immediate threats).

SETTLERS OF CATAN has at its core a dynamic engine affected by chance (see figure B.5). The roll of the dice which game board tiles will produce resources at the start of each player's turn. The more villages the player builds the more chance that the player will receive resources every turn. The player can also upgrade villages into cities which doubles the resource output of each tile. SETTLERS OF CATAN gets around the typical signature a dynamic engine creates by allowing different types of invest actions, and using a measure of upgrades instead of energy to determine the winner.

Related Patterns

The dynamic friction and attrition patterns are good ways to balance a dynamic engine.

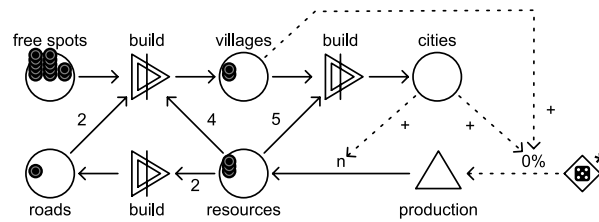


Figure B.5: The production mechanism in SETTLERS OF CATAN.

B.3 Converter Engine

Type

Engine

Intent

Two converters set up in a loop create a surplus of resources that can be used elsewhere in the game.

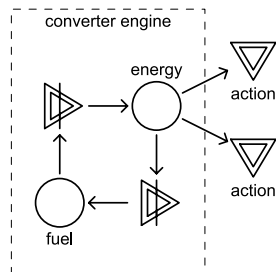
Motivation

Two resources that can be converted into each other fuel a feedback loop that produces a surplus of resources. The converter engine is a more complicated mechanism than most other engines, but also offers more opportunities to improve the engine. As a result a converter engine is nearly always dynamic.

Applicability

- Use the converter engine when you want to create a delicate mechanism to provide the player with resources. It increases the difficulty of the game as the strength and the required investment of the feedback loop are more difficult to assess.
- Use the converter engine when you need multiple options and mechanics to tune the signature of the feedback loop that drives the engine, and thereby the stream of resources that flow into the game.

Structure



Participants

- Two resources: **energy** and **fuel**.
- A **converter** that changes fuel into energy.
- A **converter** that changes energy into fuel.
- **Actions** that consume energy.

Collaborations

The converters change energy into fuel and fuel into energy. Normally the player ends up with more energy than the player started with.

Consequences

A converter engine introduces the chances of a deadlock, when both resources dry out, the engine stops working. Players run the risk of creating deadlocks themselves by forgetting to invest energy to get new fuel. Combine a converter engine with a weak static engine to prevent this from happening.

A converter engine requires more work from the player, especially when the converters need to be operated manually.

As with dynamic engines, a positive feedback loop drives a converter engine, in most cases this feedback loop needs to be balanced by applying some sort of friction.

Implementation

The number of steps involved in the feedback loop of a converter engine for a large part determines its difficulty to operate efficiently: more steps increase the difficulty, fewer steps reduce the difficulty. At the same time the number of steps also positively affects the number of opportunities for tuning or building the engine.

With too few steps in the system, the advantages of the converter engine are limited and one might consider replacing it with a dynamic engine. Too many

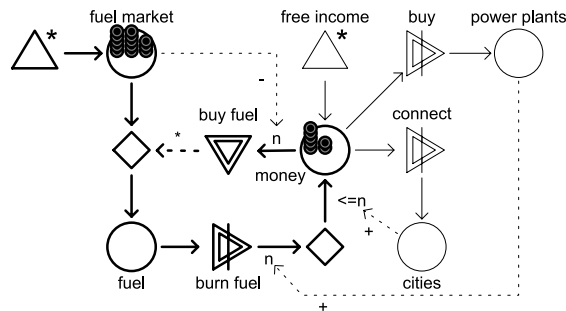


Figure B.6: The production mechanism in POWER GRID. The converter engine is printed bolder.

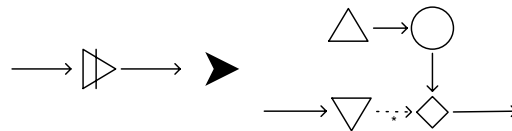


Figure B.7: Elaboration of a converter used in POWER GRID.

steps might result in an engine that is cumbersome to operate and/or maintain, especially in a board game where the different elements of the engine usually cannot be automated.

It is possible to create an unpredictable converter engine by introducing randomness, multiplayer dynamics or skill into the feedback loop. This complicates the converter engine further and often increases the chances of deadlocks.

Many implementations of the converter engine pattern put a limiter gate somewhere in the cycle in order to keep the positive engine under control and to keep the engine from producing too much energy. For example, if the number of fuel resources that can be converted each turn is limited the maximum rate at which the engine can run is capped.

Examples

A converter engine is at the heart of POWER GRID (see figure B.6). Although one of the converters is replaced by a slightly more complex construction (see figure B.7). The players spend money to buy fuel from a market, and use that fuel to generate money in power plants.¹ The surplus money is invested in more efficient power plants and connecting more cities to the player's power network. The converter engine is limited: the player can only earn money for every connected city, which effectively caps the energy output during a turn, and provides another opportunity for dynamic engine building. POWER GRID also has a weak static engine to prevent deadlocks: the player will collect a little

¹The fiction of the game is that players generate and sell electricity, but as electricity is not a tangible resource in the game it is left out of this discussion.

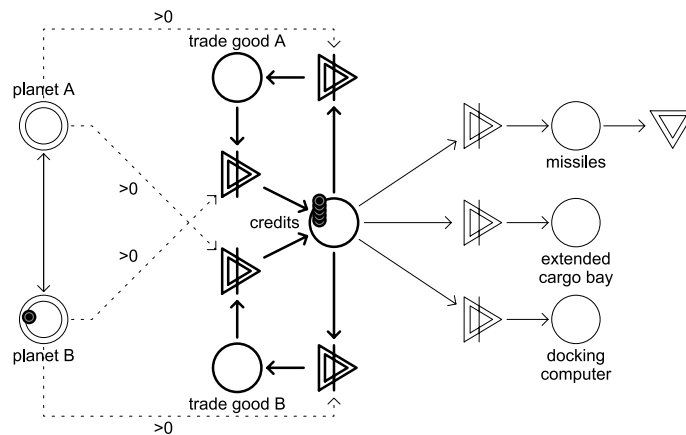


Figure B.8: Travel and trade in ELITE. The player's location on planet A or planet B activates the converters that implement the trading mechanisms in the center. A few possible ship upgrades are included on the right.

amount money during a turn even if the player failed to generate money through power plants. The converter engine of POWER GRID is slightly unpredictable as players can drive up to price of fuel by stockpiling it, which acts as a stopping mechanism at the same time.

The eighties computer space-trading game ELITE features an economy that occasionally acts as a converter engine. In ELITE every planet has its own market selling and buying various trade goods. Occasionally players will discover a lucrative trade route where they can buy one trade good at planet A and sell it at a profit planet B, and return with another good which is in high demand on planet A again (see figure B.8). Sometimes these routes involve three or more planets. Essentially such a route is converter engine. It is limited by the cargo capacity of the player's ship, which for a price can be extended. Other properties of the player's ship might also affect the effectiveness of the converter engine: the ship's "hyperspeed" range, and also its capabilities (or cost) to survive a voyage through hostile territories all affect the profitability of particular trade routes. Eventually trade routes become less profitable as the player's efforts reduce the demand, and thus the price, for certain goods over time (a mechanism that is omitted from the diagram).

Related Patterns

A converter engine is well suited to be combined with the engine building pattern as it has many opportunities to change settings of the engine: the conversion rate of two converters and possibly the setting of a limiter.

A converter engine is best balanced by introducing some sort of friction or some other form of negative feedback.

B.4 Engine Building

Intent

A significant portion of gameplay is dedicated to building up and tuning an engine to create a steady flow of resources.

Motivation

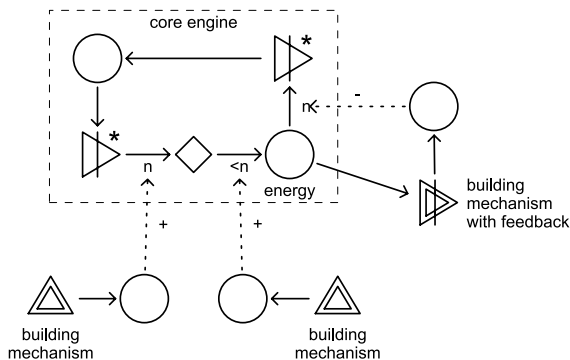
A dynamic engine, converter engine or a combination of different engines form a complex and dynamic core of the game. The game includes at least one, but preferably multiple, mechanics to improve the engine. These mechanics can involve multiple steps. It is important that assessing the current state of the engine is no trivial task.

Applicability

Use engine building when you want to create a game that:

- has a strong focus on building and construction.
- focuses on long-term strategy and planning.

Structure



The structure of the Core Engine is an example. There is no fixed way of building the engine. Engine building only requires that several building mechanisms operate on the engine and that the engine produces energy.

Participants

- The **core engine** usually is a complex construction combining multiple engine types.
- At least one, but usually multiple **building mechanisms** to improve the core engine.
- **Energy** is the main resource produced by the core engine.

Collaborations

Building mechanisms increase the output of the engine. If energy is required to activate building mechanisms then a positive, constructive feedback loop is created.

Consequences

Engine Building increases the difficulty of a game, it is best suited for lower paced games as it involves planning and strategic decisions.

Implementation

Including some form of unpredictability is a good way to increase difficulty, generate varied gameplay and avoid dominant strategies. Engine building offers many opportunities create unpredictability as the core engine tends to consist of many mechanisms. The complexity of the core engine itself usually also causes some unpredictability.

When using the engine building pattern with feedback it is important to make sure the positive, constructive feedback is not too strong and not too fast. In general, you want to spread out the process of engine building over the entire game.

An engine building pattern operates without feedback when energy is not required to activate building mechanisms. This can be viable structure when the engine produces different types of energy that affect the game differently, and allows the players to follow strategy that favor particular forms of energy above others. However, it usually does require that the activation of building mechanisms is restricted in some way.

Examples

SIMCITY is a good example of engine building. The energy in SIMCITY is money which is used to activate most building mechanisms, which in the game consists of preparing building sites, zoning, building infrastructure, building special buildings, and demolition. The core engine of SIMCITY is quite complex with many internal resources such as people, job vacancies, power, transportation capacity, and three different types of zones. Feedback loops within the engine cause all sorts of friction and effectively balance the main positive feedback loop, up to the point that the engine can more or less collapse if the player is not careful and manages the engine well.

In the board game PUERTO RICO each player builds up a New World colony. The colony generates different types of resources that can be reinvested into the colony or converted into victory points. The core engine includes many elements and resources such as plantations, buildings, colonists, money and a selection of different crops. PUERTO RICO is a multiplayer game in which they players compete for a limited number of positions which allow different actions

to improve the engine; they compete for different building mechanics. This way a strong multiplayer dynamic is created that contributes much of its gameplay.

Most real-time strategy games follow a complex engine building pattern combined with the attrition pattern.

SETTLERS OF CATAN follows an engine building structure that uses feedback and multiple types of energy. In this particular case the energy produced by the engine is never used outside the engine and its building mechanisms. This works in SETTLERS OF CATAN because the objective of the game is to score a number of points that are directly derived from the number of upgrades to the engine.

Related Patterns

Applying multiple feedback to the building mechanisms is a good way to increase the difficulty of the engine building pattern.

All friction patterns are suitable to balance the possible positive feedback that is created by an implementation of engine building that consumes energy to activate building mechanisms.

B.5 Static Friction

Type

Friction

Intent

A drain automatically consumes resources produced by the player.

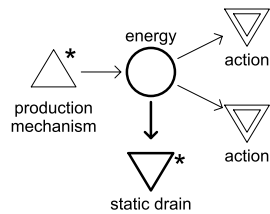
Motivation

The static friction pattern counters a production mechanism by periodically consuming resources. The rate of consumption can be constant or subjected to randomness.

Applicability

- Use static friction to create a mechanism that counters production, but which can eventually be overcome by the players.
- Use static friction to exaggerate the long-term benefits from investing in upgrades for a dynamic engine.

Structure



Participants

- A resource: **energy**.
- A **static drain** that consumes energy.
- A **production mechanism** that produces energy.
- Other **actions** that consume energy.

Collaborations

The production mechanism produces energy players need to use to perform actions. The static drain consumes energy outside players direct control.

Consequences

The static friction pattern is a relative simple way to counter positive feedback created by engine patterns. Although it tends to accentuate the typical gameplay signature of the dynamic engine because it makes upgrades of the static engine more valuable as the friction is not affected by the number of upgrades, current energy levels, or players' progress.

Implementation

An important consideration when implementing static friction is whether the consumption rate is constant or subjected to some sort of randomness. Constant static friction is the easiest to understand and most predictable, whereas random static friction can cause more noise in the dynamic behavior in the game. The latter can be good alternative to using randomness in the production mechanism. The frequency of the friction is another consideration: when the feedback is applied at short intervals the overall system will be more stable than when the feedback is applied at long or irregular intervals, which might cause periodic behavior in the system. In general, the effects of a gradual loss of energy on the dynamic behavior of the system is less than the same amount of energy is lost periodically.

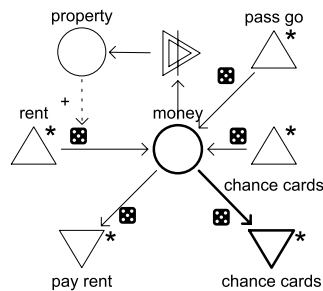


Figure B.9: Static friction in MONOPOLY.

Examples

In the Roman city building game CAESAR III the player needs to pay tributes to the emperor at particular moments during each mission. The schedule of the tributes is fixed for each mission and not affected by the player's performance. In effect, they cause uses a very infrequent, and high form of static friction, that causes a huge tremor in the game's internal economy. See section 6.3 for a more detailed discussion of this game.

The dynamic engine in MONOPOLY is countered by different types of friction, including static friction (see figure B.9). The main mechanism that implements static friction is the chance cards through which the player infrequently loses money. Although some of these cards take into account the player's property, most of them do not. One could also argue that paying rent to other players is also a form of static friction, as the frequency and severity is far beyond the direct control of the player. But, the rate of the friction does change over time and players have some indirect effect on it: when a player is doing well, chances are that the opponents are not, which negatively impacts this friction. The diagram in figure B.9 does not show this as it is made from the scope of an individual player. This type of friction is an example of the attrition pattern.

Related Patterns

As static friction exaggerates long-term investments it is best suited to be used with combination with a static engine, converter engine, or an engine building pattern.

B.6 Dynamic Friction

Type

Friction

Intent

A drain automatically consumes resources produced by the player, the consumption rate is affected by the state of other elements in the game.

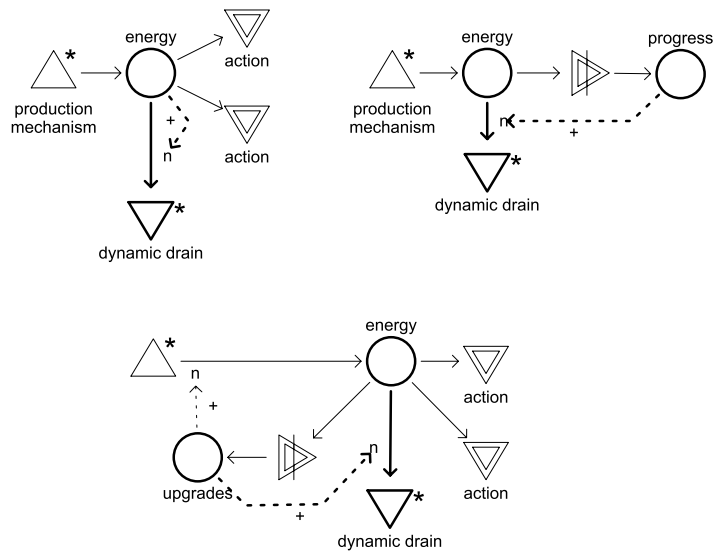
Motivation

Dynamic friction counteracts production but adapts to the performance of the player. Dynamic friction is a very typical application of negative feedback in a game.

Applicability

- Use dynamic friction to balance games where resources are produced too fast.
- Use dynamic friction to create a mechanism that counters production that automatically scales with players' progress or power.
- Use slow dynamic friction to reduce the effectiveness of long term strategies created by a dynamic engine in favor for short term strategies.

Structure



Participants

- A resource: **energy**.
- A **dynamic drain** that consumes energy.

- A **production mechanism** that produces energy.
- Other **actions** that consume energy.

Collaborations

The production mechanism produces energy players need to use to perform actions. The dynamic drain consumes energy outside players direct control, but affected by the state of at least one other element in the game system.

Consequences

Dynamic friction is a good way to counter positive feedback created by engine patterns. Dynamic friction adds a negative feedback loop to the game system.

Implementation

There are several ways of implementing dynamic feedback. An important consideration is the choice of the element that causes the consumption rate to change. In general, this can either be the energy itself, the number of upgrades to a dynamic engine or a converter engine, or the player's progress towards a goal. When energy is the cause for the friction to change, the negative feedback tends to be fast, when progress or production power is the cause the feedback is more indirect, and probably slower.

When dynamic friction is used to counter a positive feedback loop it is important to consider the difference in characteristics of the positive feedback loop and the negative feedback loop implemented through the dynamic friction. When the characteristics are similar (equally fast, equally durable, etc.), the effect is far more stable than when the differences are large. For example, when a slow and durable dynamic friction is acting against a fast but non-durable positive feedback that initially yields a good return, players will initially make a lot of progress, but might suffer in the long run. Fast positive feedback and low negative feedback seems to be the most frequently encountered combination.

Examples

Dynamic friction is used the city production mechanism in *CIVILIZATION* (see figure B.10). In this game the player builds cities to produce food, shields and trade. As cities grow they need more and more food for their own population. Players have some control over how much food is produced compared to other resources, but are limited in their options by the surrounding terrain. By choosing to produce a lot of food cities initially produce fewer other resources, but grow faster to great potential. Fast growth creates a problem however, the happiness rating a city must stay equal to or higher than half the population, else the production stops because of civil unrest. Initially a city has a happiness value of 2. Players can create more happiness by building special buildings, or by converting trade into culture. Both ways cause more dynamic friction with different

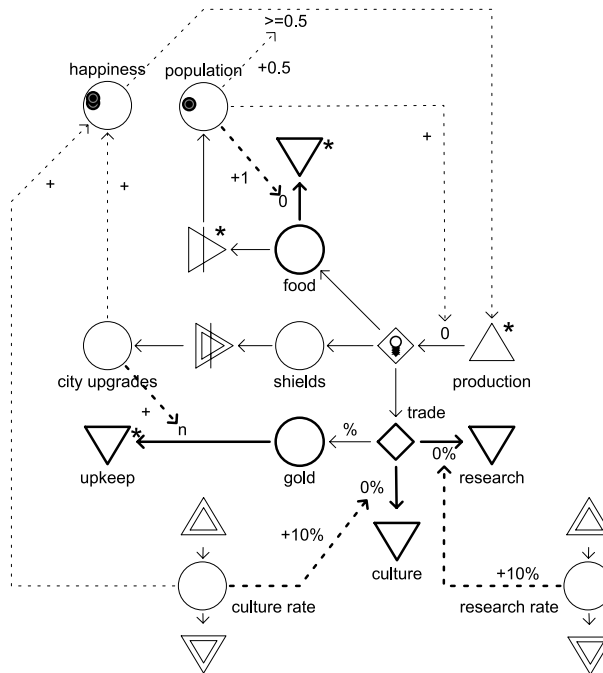


Figure B.10: The city economy of CIVILIZATION. Dynamic friction is printed bold. The player can freely adjust the culture and research settings in order to control unrest and research production. These settings are global and affect all cities equally.

profiles on the production process; the first is slow, requires a high investment, but is highly durable, and has relative high return, whereas the second is fast, but has a relative low return for its investment.

Related Patterns

Dynamic friction is a good way to balance any pattern that causes positive feedback, and often is part of the multiple feedback pattern.

B.7 Attrition

Type

Friction

Intent

Players actively steal or destroy resources of other players that they need for other actions in the game.

Also Known As

Multiplayer, destructive feedback

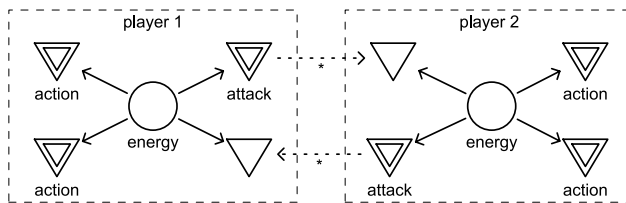
Motivation

By allowing player to directly steal or destroy each others resources, players can eliminate each other in a struggle for dominance.

Applicability

- Use attrition to allow direct and strategic interaction between multiple players.
- Use attrition to introduce feedback into a system which nature is determined by the strategic preferences or whims of the players.

Structure



Participants

- Multiple **players** that have the same (or similar) mechanics and options.
- A resource used by all players: **energy** to perform **actions**.
- A special **attack** action that triggers or activates a drain of other player's energy.

Collaborations

By performing attack actions players can drain each other's energy. Attacking typically costs some sort of energy too, or at least attacking players can spend less time on other actions. The balance between the attack costs, how much energy players lose to an attack, and how beneficial the other actions are, determine the effectiveness the attack and the pattern.

Consequences

Attrition introduces a lot of dynamism into a system as players directly control the measure of the destructive force applied to each other. Often this introduces destructive feedback as the current state of a player will cause reactions by other

players. Depending on the nature of the winning conditions and the current state of the game this feedback might be negative when it stimulates players to act and conspire against the leader, but it also might cause positive feedback when players are stimulated to attack and eliminate weaker players.

Implementation

For attrition to work it is best that players must invest some sort of resource into attack that could also be spend otherwise. Without this investment attrition simply becomes race to destroy each other with little or no strategic choices. Although this might work when there are multiple players are involved and the game facilitates social interaction between players. In this case the players need to chose whom to attack.

It is quite common to implement attrition using two resources to represent life and energy. In this case players use energy to perform actions and loose when they run out of life. When using these two resources it is important that they are somehow related, often players are allowed to spend energy to gain more life. Sometimes the relation between life and energy is implicit, for example when a player must choose between spending energy or gaining life, there is an implicit link between the two as players generally cannot do both at the same time.

In a two player version of attrition the game must include other actions, and games for more than two players often also allow other actions to be performed by the player. Most of the time a these actions constitute some sort of production mechanism for energy, increases the effectiveness the players defensive or offensive capabilities. Most real-time-strategy games include all these options, often with multiple variants for each.

The winning conditions and effects of eliminating another player have a big impact on the attrition pattern. The winning condition does not need be the elimination of players, players might score points, or reach a particular goal outside the attrition pattern which automatically widens the number of strategies involved. When there is a bonus to attacking or eliminating players, the pattern can be made to stimulate weaker players.

Examples

Almost all strategy game implement some sort of attrition as it is often an important goal to eliminate other players in this type of games. The SIMWAR example, discussed in Chapter 4, provides a good example.

The trading card game MAGIC THE GATHERING implements a decorated version of the attrition pattern. Figure figure B.11 presents this implementation, although it show the details from the perspective of a single player only. In MAGIC THE GATHERING, players can play one card every turn, these cards allow players to add lands, summon creatures or cast spells to heal, or deal direct damage to their opponent or their opponent's creatures. But all actions, except playing lands costs mana, the more mana players have the more they can spend each turn and the more powerful actions they can play. Creatures will fight

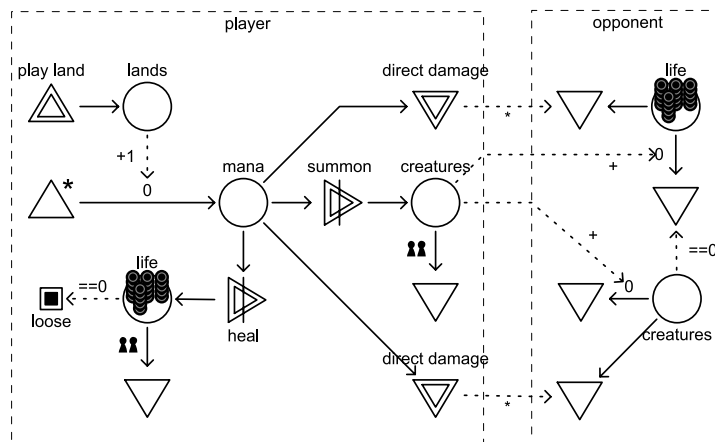


Figure B.11: The attrition mechanism in MAGIC THE GATHERING.

other creatures and when there are no more enemy creatures they will damage the opponent directly. Players that lose all their life points are eliminated from the game. gameMagic the Gathering is an example of a game that implements attrition using separate resources for life and energy (in this case life and mana). The different options illustrate how attrition can work differently: direct damage activates a trigger briefly. As its name implies, it is fast and direct. On the other hand, summoning creatures activates a permanent drain on the opponent's creatures and life. The effects usually are not as powerful as direct damage, but as they accumulate over time they can be quite devastating. What options are available to a player and how powerful these options are exactly, is determined by the cards in the player's hand. As players build their own decks from a large collection of cards, deck building is an important aspect of MAGIC THE GATHERING.

The most obvious way to implement attrition is in a symmetrical game. However, many single-player games and even certain types of multiplayer games use 'a-symmetrical' attrition. An example of 'a-symmetrical' attrition can be found in the board game SPACE HULK where one player, controlling a handful of 'space marines', tries to accomplish a mission while the other player, controlling an unlimited supply of alien 'Genestealers' tries to prevent that. The genestealer player tries to reduce the number of space marines to stop them from accomplishing their goals, they win when they have destroyed enough space marines. On the other hand, the space marines usually cannot win by destroying a certain number of genestealers, but they must keep the number of genestealers under control in order to survive, as the genestealers become more effective as their numbers grow. Figure B.12 is a rough illustration of these mechanics in SPACE HULK.

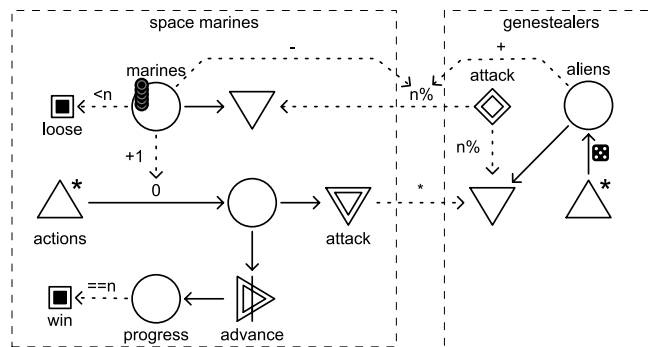


Figure B.12: A-symmetrical attrition in SPACE HULK.

Related Patterns

Attrition works well with any sort of engine pattern. Trade can be used as an alternative form of multiplayer feedback that is constructive instead of destructive, and nearly always negative.

B.8 Stopping Mechanism

Type

Friction

Intent

Reduce the effectiveness of a mechanism every time it is activated.

Motivation

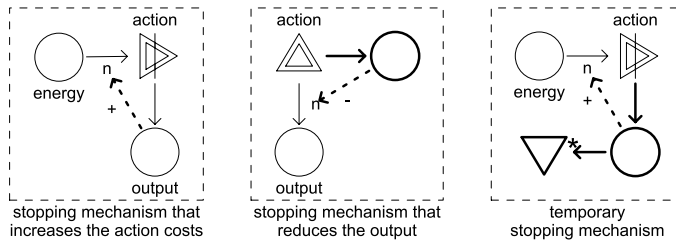
In order to prevent a player from abusing a powerful mechanism its effectiveness is reduced every time it is used. In some cases the stopping mechanism is permanent, but usually its not.

Applicability

Use stopping mechanism to:

- prevent players from abusing particular actions.
- counter dominant strategies
- reduce the effectiveness of a positive feedback mechanism.

Structure



Participants

- An **action** that might produce some sort of **output**.
- A resource **energy** that is required for the action.
- The **stopping mechanism** that increases the energy cost or reduces the output of the action.

Collaborations

For a stopping mechanism to work it the action must have an energy cost, an output, or both. The stopping mechanism reduces the effectiveness of an action mechanism every time it is activated by increasing the energy costs or reducing the output.

Consequences

Using a stopping mechanism can reduce the effect a positive feedback loop considerably and even make its return insufficient.

Implementation

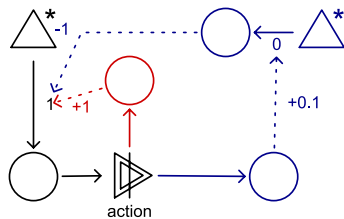
When implementing a stopping mechanism is important to consider whether or not to make the effects permanent or not. When the accumulated output is used to measure the strength of the stopping mechanism the effects are not permanent, in that case it requires players to alternate frequently between creating the output and using the output in other actions.

A stopping mechanism can apply to each player individually or can affect multiple players equally. In the latter case, players will reward players that use the action before other players do. This means that the stopping mechanism can create a form of feedback depending on whether leading or trailing players are likely to act first.

Examples

The price mechanic of the fuel market in POWER GRID involves a stopping mechanism (see figure B.13). In POWER GRID players use money to buy fuel

Structure



In the example structure above there are two feedback mechanisms. The action (black) activates one feedback mechanism (red) which is positive, fast and direct, but it also activates a secondary feedback mechanism (blue) which is negative, slow and indirect. This illustrates just one way of setting up multiple feedback loops. There are many more.

Participants

- An **action** that can be activated by the player.
- Multiple **feedback mechanisms** that are activated by the action.

Collaborations

The action activates multiple feedback mechanisms that ultimately feedback into the action.

Consequences

For the player multiple feedback loops are more difficult to understand than single feedback loops. As a result using this pattern makes a game more difficult.

If the feedback loops the action activates can have dynamic signatures that change during play (which they often have) it is very important for the player to be able to read the current signature, as their balance might shift considerably during the game.

Finding the right balance between the multiple feedback loops is an important issue in a game that uses this pattern.

Implementation

When creating a game with multiple feedback it is very important to make sure that the signatures of the different feedback loops are different. In particular the speed of the feedback needs to vary if this pattern is going to be effective. Alternatively varying the signature of the feedback over time can also work well. To this end adding playing style reinforcements and stopping mechanisms to one or more of the feedback loops is a good design strategy.

The most common combination for multiple feedback seems to be fast, constructive, positive feedback coupled with slow, negative feedback. This creates a trade-off between short term gains and long term disadvantages.

Examples

Attacking in RISK feeds into three positive feedback loops of varying speeds and strengths. The most obviously using armies to capture more lands allows the player to build more armies. A slower form of feedback is realized by the cards: a player that successfully attacks gains a card, certain combination of three cards allow him to gain extra armies. The last type of feedback is created by the capturing of continents. A continent will give a player a number of bonus armies each turn, this is very fast and strong feedback loop, but one that requires a higher investment of the player.

Related Patterns

Playing style reinforcements and stopping mechanisms are good ways to ensure that the signature of the feedback loops an action feeds into change over time.

B.10 Trade

Intent

Allow trade between players to introduce multiplayer dynamics and negative, constructive feedback.

Motivation

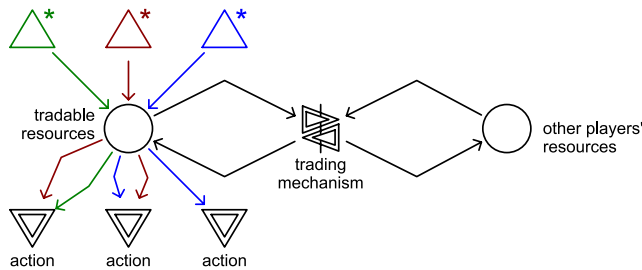
Players are allowed to trade important resources. Usually this means that leading players will get tougher deal while trailing players can help each other to catch up. Trade works especially well when the flow of resources is unstable and/or not equal distributed among players.

Applicability

Use trade to:

- introduce multiplayer dynamics to the game.
- introduce negative, constructive feedback.
- introduce a social mechanic that involves players outside their own turns.

Structure



Participants

- A **trading mechanism** that allows trade of resources.
- Multiple **tradable resources** that can be spent in various ways.
- **Actions** that require the use of tradable resources.

Collaborations

The tradable resources can be exchanged by the players using the trading mechanism.

Consequences

Trade introduces negative feedback that does not really slow down the game but usually helps trailing players catch up, (as it is not destructive).

Trade favors players with good social and bartering skills.

Implementation

In board games trade is very easy to implement, the game simply needs to specify how and when players can trade resources. In a multiplayer computer game trade is also easy. However, creating a trading mechanism that involves artificial players, is far from trivial.

To implement a successful trading mechanism, multiple tradable resources are required, and the production rates of these resources must fluctuate or at least be different among players. Trading only works when there is an imbalance in the distribution of resources among the trading parties. It also helps to include many actions that consume the tradable resources and to create actions that consume resources of multiple types, as this further exaggerates the imbalance when players choose different courses of action.

Examples

In *SETTLERS OF CATAN* players build up an uncertain dynamic engine: villages and cities that produce the resources used to build more villages and cities. The

randomness of these engines is partly countered by allowing all players to trade resources with the player that is taking his or her turn. The exchange rate is set by in mutual agreement and usually determined by the availability, accessibility of the resource and the position of the player. Players that are in the lead can afford to pay more for their resources. When close to winning players might find it impossible to make a deal.

In *CIVILIZATION III* players can exchange strategic resources, money, and knowledge. This mutually benefits both parties and allows weaker civilizations to catch up rather quickly.

Related Patterns

Attrition can be an alternative source of multiplayer feedback that is not constructive but destructive in nature.

B.11 Escalating Complications

Type

Escalation

Intent

Player progress towards a goal increases the difficulty of further progression.

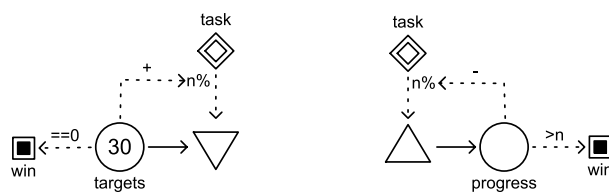
Motivation

A positive feedback loop on the game's difficulty makes the game increasingly harder for players as they get closer to achieving their goals. This way the game quickly adapts to the player's skill level, especially when the good performance allows the player to progress more quickly.

Applicability

Escalating complications is well suited for fast-paced games based on player skill where the game needs to adjust quickly to the player's skill level.

Structure



Participants

- **Targets** represent unresolved tasks.
- Alternatively, **progress** represents the player's progress towards a goal.
- A **task** that either reduces the number of targets or produces progress.
- A feedback mechanism that makes the game more difficult as the player progresses towards the goal or reduces the number of targets.

Consequences

Escalating complications is based on a simple positive feedback loop affecting the difficulty of the game. It is mechanism quickly adjusts the difficulty of the game to the skill level of the player. If failure of the task ends the game escalating complication ensures a very quick game.

Implementation

The task in a game that only implements the escalating complication pattern is typically affected by player skill, especially when the escalating complications is makes up the most of a game's core mechanics. When the task is a random or deterministic mechanic, players will have no control over the game's progress. Only when the escalating complication pattern is part of a more complex game system, where players have some sort of indirect control over the chance of success, a random or deterministic mechanic becomes viable. Using multiplayer dynamic mechanisms is an option but probably works better in a more complex game system as well.

Examples

SPACE INVADERS is a classic example of the escalating complication pattern. In SPACE INVADERS the player needs to destroy all invading alien before they can reach the bottom of the screen. Every time the player destroys an alien all other aliens speed up a little, making it more difficult for the player to shoot them.

PAC-MAN is another example. In PAC-MAN the task is to eat all the dots in a level, while the chasing ghosts further increase the difficulty of the task and adding to strategic decision to eat an 'energizer' to get rid of them.

Related Patterns

By combining static friction or dynamic friction with escalating complications a game can be created that quickly matches its difficulty to the ability of the player.

B.12 Escalating Complexity

Type

Escalation

Intent

Players act against growing complexity, trying to keep the game under control until positive feedback grows to strong and the accumulated complexity makes them loose.

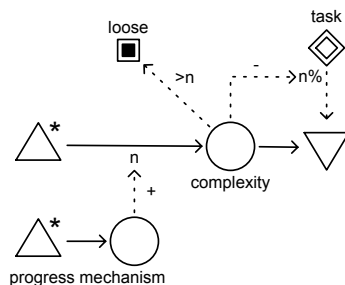
Motivation

Players are tasked to perform an action that grows more complex if the players fail and which complexity contributes to the difficulty of the task. As long as players can keep up with the game they can keep on playing, but once the positive feedback spins out of control, the game ends quickly. As the game progresses the mechanism that creates the complexity gains speeds up ensuring that at one point players can no longer keep up and eventually must loose the game.

Applicability

Use escalating complexity when you aim for an addictive skill based game.

Structure



Participants

- The game produces complexity that must be kept under under a certain limit by the player.
- The task is a player action that reduces complexity.
- Progress mechanism increases the production of complexity over time.

Collaborations

Complexity immediately increases the production of more complexity, creating a strong positive feedback loop that must be kept under control. The player loses when complexity exceeds a particular limit.

Consequences

Given enough skill players can keep up with the creation of complexity for a long time, but when players no longer keeps up complexity spins out of control and the game ends quickly.

Implementation

The task in a game that implements the escalating complexity pattern is typically affected by player skill, especially when escalating complexity makes up most of the game's core mechanics. When the task is a random or deterministic mechanic, players will have no control over the game's progress. Only when the escalating complexity pattern is part of a more complex game system, where players have some sort of indirect control over the chance of success, a random or deterministic mechanic becomes viable. Using a multiplayer dynamic task is an option but probably also works better in a more complex game system.

Randomness in the production of complexity creates a game with a varied pace, where players might struggle keeping up with a peek in production before catching some breath when production dies out a little.

There are many ways to implement the progress mechanic, from a simple time based increase of the production of complexity (as is the case in the sample structure above) to complicated constructions that rely on other actions of the player or other player actions. This way it is possible to combine escalating complexity with escalating complication by introducing positive feedback to the progress mechanic as a result of the execution of the task.

Escalating complexity lends itself well to combine with a multiple feedback structure where the complexity feeds into several feedback loops with different signatures. For example, escalating complexity can be partially balanced by having the task feeding into a much slower negative feedback loop on the production of complexity.

Examples

In TETRIS a steady flow of falling bricks causes complexity. There is a slight randomness in this production as the different blocks are created all the time. Player need to place the blocks in such way that they fit together closely. When a line is completely filled it disappears making room for new blocks. When players fail to keep up the blocks pile up quickly and they will have less time to place subsequent blocks, this can quickly increase the complexity of the field when players are not careful and causes them to loose the game if the pile of

blocks reach the top of the screen. The game progressively speeds up by making the blocks fall faster and faster, making it more and more difficult.

Related Patterns

Any type of engine pattern can be used to implement the progress mechanism.

B.13 Playing Style Reinforcement

Intent

By applying slow, positive, constructive feedback on player actions, the game gradually adapts to the players preferred playing style.

Also Known As

RPG-elements (as in ‘a game with RPG-elements’)

Motivation

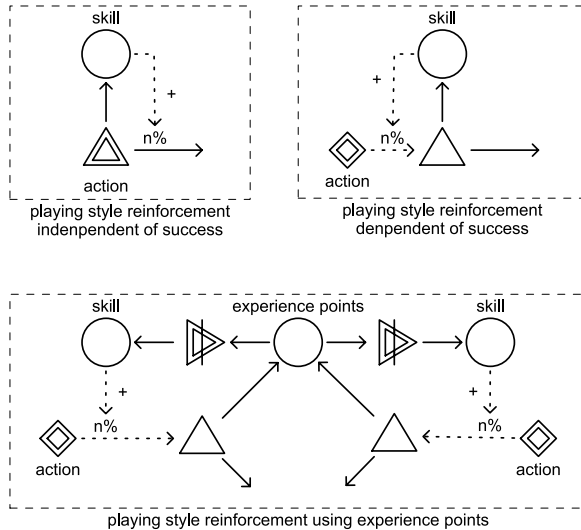
Slow, positive, constructive feedback on players’ actions that also have another game effect causes the players’ avatar or units to develop over time. As the actions themselves feed back into this mechanism the avatar or units specialize over time, getting better in a particular task. As long as there are multiple viable strategies and specializations, the avatar and the units will, over time, reflect the player’s preferences and style.

Applicability

Use Playing Style Reinforcement when:

- you want players to make a long-term investment in the game that spans multiple sessions.
- you want to reward players for building, planning ahead, and developing personal strategies.
- you want players to grow into a specific role or strategy.

Structure



Participants

- **Actions** players can perform which success is not completely dependent on the players skill.
- A resource **ability** that affects the chance actions succeed and than can grow over time.
- An optional resource Experience points that can be used to increase an ability.

Collaborations

Ability affects the success rate of actions.

Performing actions generates experience points or directly improves abilities. Some games require the action to be successful, while others do not.

Experience points can be spent to improve abilities.

Consequences

Playing style reinforcement works best in games that are played over multiple sessions and over a long time.

Playing style reinforcement only works well when multiple strategies and play styles are viable option in the game. When there is only one, or only a few, every one will go for those options.

Playing style reinforcement can inspire min-maxing behavior with players. With this behavior players will seek the best possible options that will allow them to gain powerful avatars or units as fast as possible. This can happen

when the strength of the feedback is not distributed evenly over all actions and strategies.

Playing style reinforcement favors experienced players over inexperienced players, as the first group will have a clearer view over all the options and long-term consequences of their actions.

Playing style reinforcement rewards the player who can invest the most time in playing the game. In this case, time can compensate for playing skill, which can be a wanted or unwanted side-effect.

It can be very ineffective to change strategies over time in a game with playing style reinforcement, as previous investments in another play style will not be used as efficiently.

Implementation

Whether or not to use experience points is an important decision when implementing play style reinforcement. When using experience points there is no direct coupling between growth and action, allowing the player to harvest experience with one strategy to develop the skills to excel in another strategy. On the other hand, if you do not use experience points you have to make sure that the feedback is balanced for the frequency of the actions: actions that are performed more often should have weaker feedback than actions that can be practiced infrequently.

Role-playing games are the quintessential games built around the play style reinforcement pattern. In these games the feedback loops are generally quite slow, and balanced by dynamic friction or a stopping mechanism to make sure avatars do not progress too fast. In fact, most of these games are balanced in such way that progression is initially fast and gradually slows down, usually because the required investment of experience points increases exponentially.

Whether or not the action needs to be executed successfully to generate the feedback is another important design decision and can inspire totally different player behavior. When success is required, the feedback loop gains influence. In that case it is probably best to have task difficulty also affect the success of an action, and to challenge the player with tasks of different difficulty levels allowing them to train their avatars. When success is not required, players have more options to improve neglected abilities during later and more difficult stages. However, it might also inspire players to perform a particular action at every conceivable opportunity, which could lead to some unintended unrealistic or comic results, especially when the action involves little risk.

Examples

In the board game BLOOD BOWL players coach football teams in a fantasy setting. Individual team members score 'star player points' for successful actions: scoring touchdowns, throwing complete passes or injuring opposing players. When a team member collected enough star player points he gains new or improved abilities. Many of these increase their ability to score, pass or injure

opponents. Improvements occur only between matches and players build up a team over a long series of matches. BLOOD BOWL facilitates a wide variety of playing styles that generally fall somewhere between two poles: agility play with a strong focus on ball handling and scoring, and strength play with a strong focus on taking out the opposition in order to win the game.

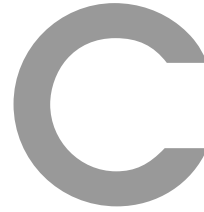
In computer role-playing game THE ELDER SCROLLS IV: OBLIVION the player avatar's progress is directly tied to their actions. The avatar's ability corresponds directly to number of times he performed the associated actions.

In CIVILIZATION III there are different ways in which a player can win the game. A player reinforces his chosen strategy of military, economic, cultural or scientific dominance (or any combination), by building city improvements and world wonders that favor that strategy. In CIVILIZATION III several resources take the role of experience points: money and production among them. These resources are not necessarily tied to one particular strategy in the game: money generated by one city can be spent to improve production in another city in the game.

In the board game CAYLUS players occasionally win a privilege. They can choose to use the privilege to gain money, points, building resources, or the opportunity to build new structures. Every time they pick one of them the option is improved (the first time you get 1 point, the second time 2 points, etc). Rules are in place to prevent a player from spending privileges on the same option to often. This is an example of playing style reinforcement without experience points.

Related Patterns

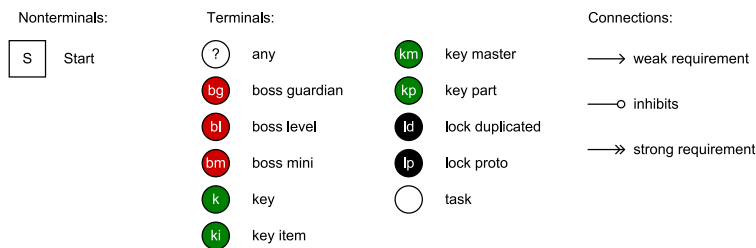
When playing style reinforcement depends on the success of actions, it creates a powerful feedback. In that case a stopping mechanism is often used to increase the price of new upgrades to an ability.



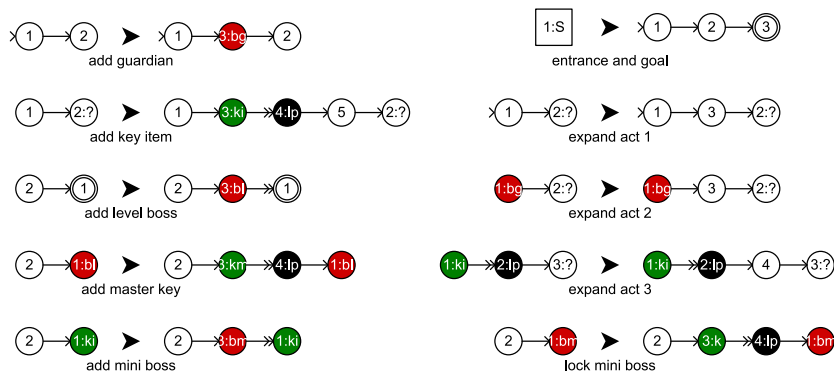
A Recipe for Zelda-esque Levels

The following 'recipe' describes the formal grammar and rewrite systems to generate levels in style of THE LEGEND OF ZELDA.

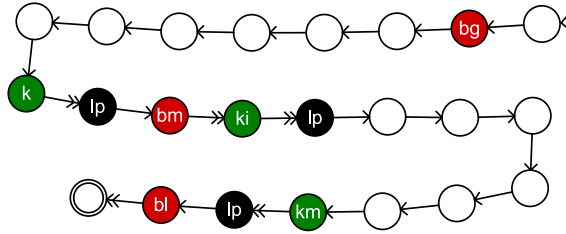
Step 1: generate basic mission



Rewrite rules:



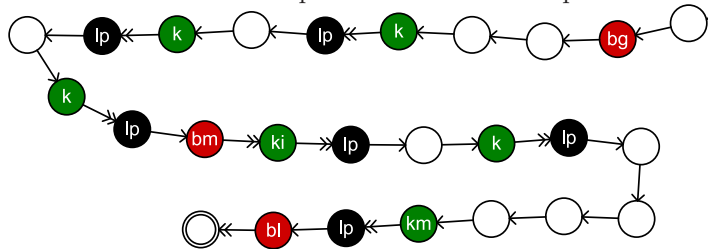
This step creates a basic mission. It applies the following rules once: entrance and goal; add key item; add level boss; add mini boss; add master key; add guardian; lock mini boss. Next it applies the rule 'expand act 2' between somewhere between 3 and 5 times. Finally it applies the rule 'expand act 3' between somewhere between 4 and 6 times. Sample outcome of this step:



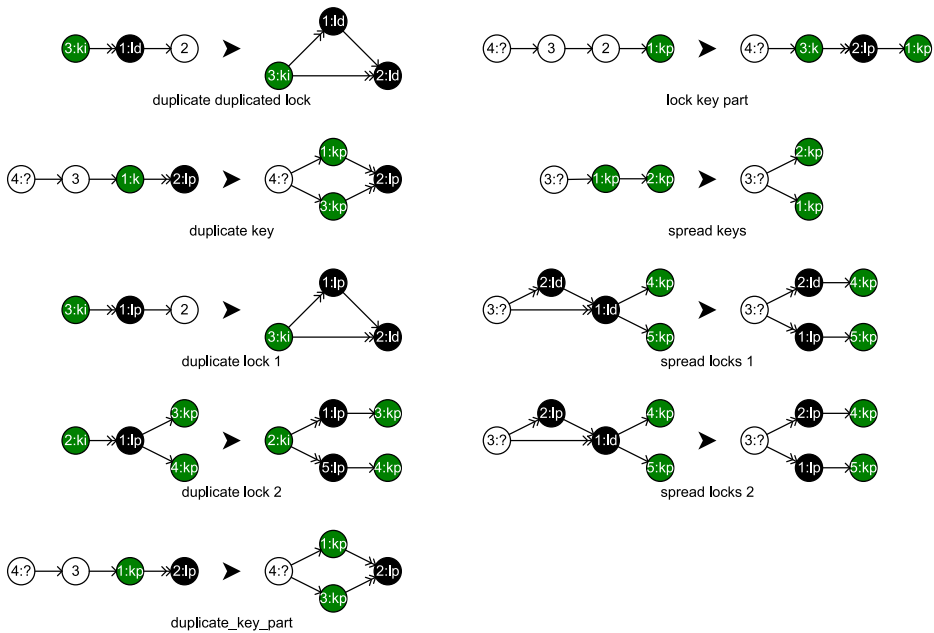
Step 2: add locks



This step adds a few extra locks to the mission. It applies its single rule between 1 and 3 times. Sample outcome of this step:

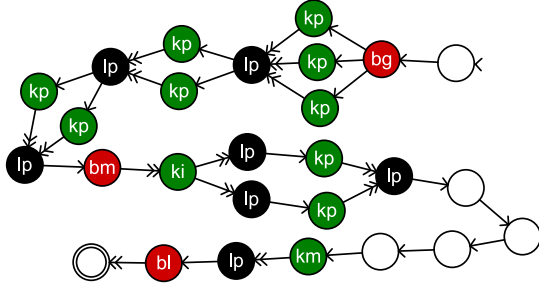


Step 3: complicate locks

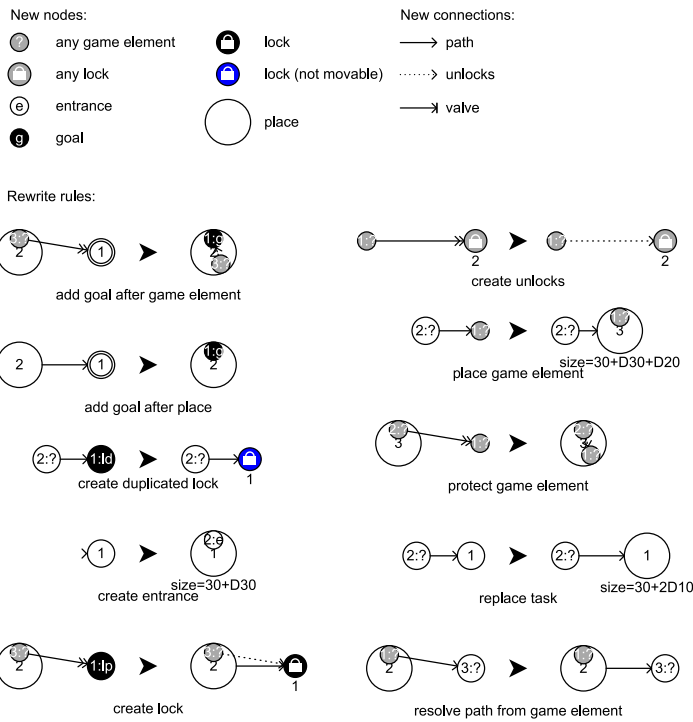


This step refines the existing locks by duplicating keys and locks. Rules are

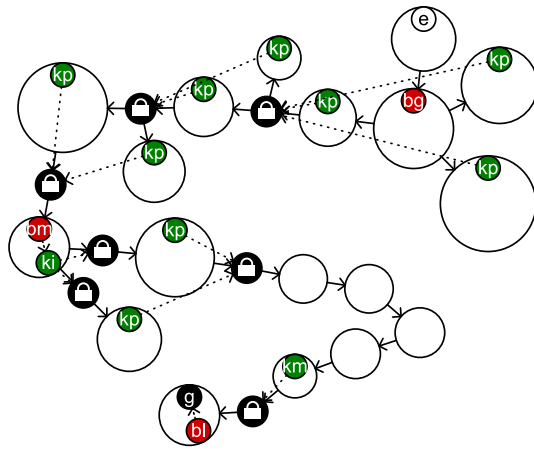
applied until all options are exhausted. Sample outcome of this step:



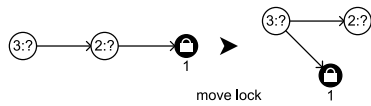
Step 4: transform into space



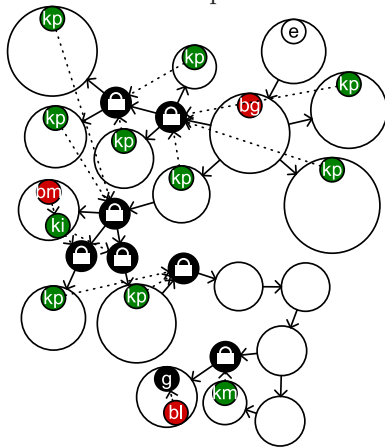
This rewrite system transforms the mission graph into an almost isomorphic space graph. Rules are applied until all options are exhausted. Sample outcome of this step:



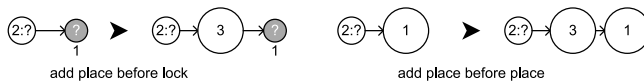
Step 5: move locks



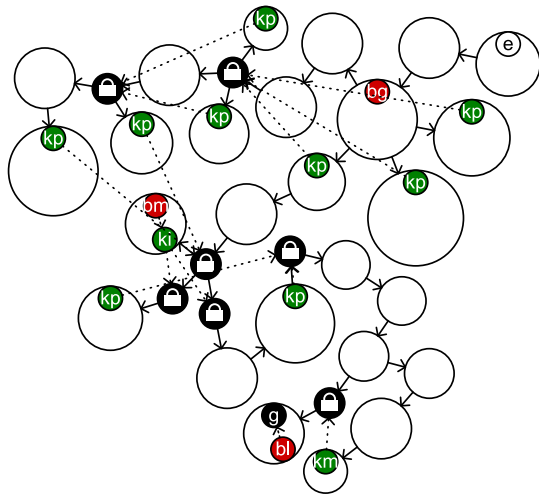
This step moves locks towards the entrance. It applies its single rule between 6 and 11 times. Sample outcome of this step:



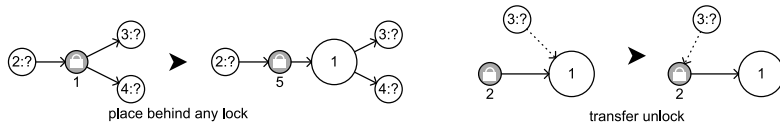
Step 6: expand space



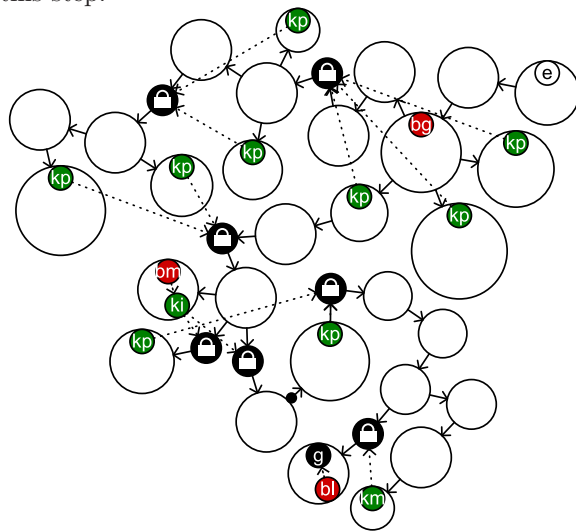
This step adds a few rooms to the structure. It applies rules randomly between 5 and 9 times. Sample outcome of this step:



Step 7: clean up locks



This rewrite systems makes sure locks are no longer directly connected to other locks. Rules are applied until all options are exhausted. Sample outcome of this step:

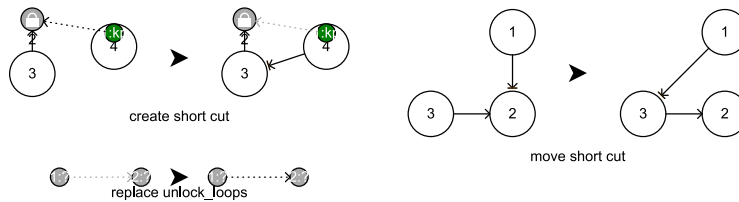


Step 8: create short cuts

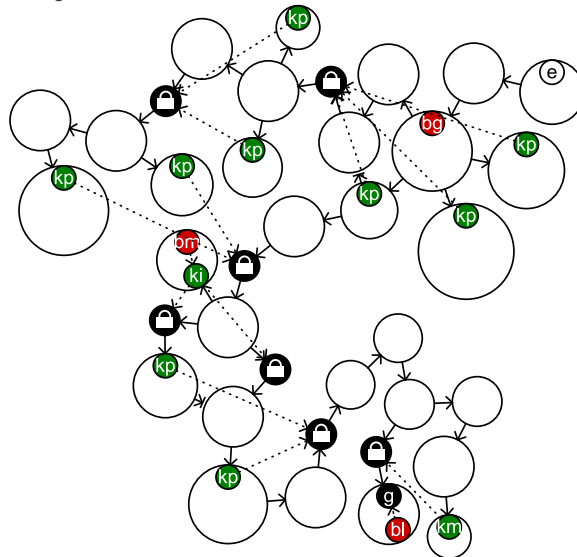
New connections:

-----> unlocks_loops

Rewrite rules:



This step introduces short cuts to allow the player to quickly return to a lock after finding a key. It applies the rule 'create short cut' between 2 and 3 times, then it applies the rule 'move short cut' between 0 and 3 times. Finally it applies the rule 'replace unlock-loops' as many times as possible. Sample outcome of this step:



Summary

Game design theories aim to assist game developers in creating and understanding games. Applied theory for game design supports creativity, speeds up the design process, and increases quality of games. This dissertation presents two theoretical frameworks for designing games and discusses how designers can use them to refine the game development process.

Currently there are several design theories for games. However, none of these theories has surfaced as a widely accepted standard within the game industry. Most existing (often academic) design theories have a poor track record. There are two reasons for this. One: they require a considerable investment to learn, and two: they are more successful as analytical tools than as actual development tools. It should come as no surprise that this has caused considerable skepticism towards design theory within the game industry.

With these concerns in mind, the frameworks presented in this dissertation were designed to be practicable and applicable. The first framework frames games as rule-based systems. It helps to identify and understand the structure of game rules in order to facilitate design. The second framework helps designers to create open, dynamic worlds that to provide players with a smooth gameplay experience. The two frameworks are not completely separate: in a good game, game worlds and rules combine into a complex, carefully balanced whole. One cannot do without the other.

The first framework that focuses on game rules is called *Machinations* and it utilizes dynamic, interactive diagrams. These diagrams model a game's internal economy that consists of the flow of vital, and sometimes abstract, resources. Resources can be of many types, for example: points, money or ammunition.

The most important structural features these diagrams capture are feedback patterns. Feedback is generally recognized to play an important role in emergent behavior in complex systems in general, and games in particular. Feedback occurs when state changes in a game element affect other elements and ultimately feed back to affect the first element. In games, feedback often creates an upward spiral. For example, in the classic board game *Monopoly* investing money in property generates more money to invest.

The *Machinations* framework distinguishes many different types of feedback: feedback can be positive or negative, constructive or destructive, fast or slow, etcetera. For designers it is important to understand the nature of the feedback loops that operate in their games. What is more, from the study of several games and their internal economy recurrent patterns emerge. These patterns codify general solutions to commonly encountered design challenges. *Machinations* diagrams are an excellent vehicle to identify and reason about these patterns.

Game designers can create *Machinations* diagrams with the software developed for this study. The main advantage of this software is that these diagrams can be run: like the games they model, *Machinations* diagrams are dynamic and interactive. This allows designers to simulate games easily and efficiently; it allows them to balance and tweak a game's internal economy for particular effects long before a prototype is built.

The second framework, Mission/Space, focuses on level design and the mechanics that control player progression through a game. Unlike existing models of level design, the Mission/Space framework capitalizes on the idea that in a game level two different structures exist. Mission graphs are used to formalize the structure of the tasks and challenges laid out for players, while space graphs are used to formalize the geometric construction of game levels. These two constructions are related, but they are not the same. Rather, the different ways missions might be mapped onto a particular space play an important role in the resulting player experience. Mechanisms for locks and keys, which are often used to prevent players from reaching certain areas in a level too quickly, are a common tool to create mission/space structures.

As with the Machinations framework, the Mission/Space framework was used to develop software tools to assist designers in creating levels. In this case, the program Ludoscope allows designers to implement mission graphs and space graphs and simulate a player's progress through a level. This way Ludoscope helps to identify potential deadlock, or other flaws, in their design.

The Machinations framework focuses on structures of emergence in games and the Mission/Space framework focuses on structures of progression. Combining the two frameworks provides leverage to investigate how emergence and progression can be combined. Many games combine elements of both. However, there also are many games where there is a considerable mismatch between the sophistication of the mechanics in their economy, space and physics on the one hand, and the mechanics to control progression on the other. By combining the Machinations framework with the Mission/Space framework, two new strategies for integrating emergence and progression surface.

The first strategy zooms in on lock and key mechanisms that rarely include feedback mechanisms. Machinations diagrams can be used to explore more sophisticated and emergent mechanics to create locks and keys, and to control player progress. In the second strategy progress itself is modeled as a resource allowing it to become part of feedback mechanisms that operate on the scale of the entire level or game.

The Machinations framework and the Mission/Space framework formalize different perspectives on games. The notion of 'model driven engineering', taken from software engineering, can unify these different perspectives into one formal approach to game development. This perspective ultimately allows us to automate certain aspects of creating games. Using formal grammars and rewrite systems it is possible to codify transformations that allow designers to generate spaces from mission, mechanics from spaces, or missions from mechanics. By dividing the development process into a series of small transformations, flexible tools can be created that can (help to) automatically generate entire game levels, and, to a certain extent, entire games. Another application of these techniques is the development of games that adapt to player performance.

To implement these techniques, the Ludoscope prototype was extended to allow designers to define such grammars and rewrite systems. This creates a potentially very powerful, generic game development tool that allows games to be designed at a high level of abstraction.

The various tools developed as part of this research are one way to validate the applicability of the theory developed in this dissertation, but it is not the only one. Through various presentations and workshop, academy peers and industry veterans were able to comment on and influence the development of the frameworks and tools. In addition, the frameworks, and Machinations in particular, were used in the game development courses taught at the Hogeschool van Amsterdam. We found that these frameworks help students to understand games and improve their designs. Mainly because they formalize important aspects of game design that are not immediately apparent or tangible, and because they facilitate students to simulate and experiment with these aspects.

In the end, designing games remains a tough challenge. The tools and theory presented in this dissertation assist game designers by codifying design lore, simulating games before they are built, and even by automating certain aspects of the design, but they can never replace the human creativity and ingenuity that goes into their design. However, I do think that these tools and theory will assist them and can elevate the art of game design to the next level.

Nederlandse Samenvatting

Game ontwerptheorieën helpen game designers bij het maken en begrijpen van hun werk. Zulke toegepaste theorieën stimuleren het creatieve proces en kunnen de ontwikkelkosten van games drukken, het ontwerpproces versnellen en de kwaliteit van de eindproducten verbeteren. Dit proefschrift presenteert twee theoretische kaders voor het ontwerpen van games en beschrijft hoe game designers deze kunnen inzetten om het game ontwerpproces te stroomlijnen.

Er bestaan op dit moment meerdere ontwerptheorieën voor games, maar geen enkele kan rekenen op een breed draagvlak binnen de game industrie. Vooral academische ontwerptheorieën hebben regelmatig een slechte reputatie. Daar zijn twee redenen voor. Ten eerste vergen ze een grote tijdsinvestering om te leren hanteren. Ten tweede zijn ze vaak geschikter als analyse- dan als ontwikkel-instrument. Het zal dan ook niemand verbazen dat elke nieuwe ontwerptheorie binnen de game industrie op enige scepsis stuit.

De ontwerptheorieën uit dit proefschrift zijn daarom ontwikkeld met het idee dat zij praktisch en makkelijk toepasbaar moeten zijn voor game designers. Het eerste theoretische kader biedt handvatten om games te bekijken en begrijpen als systemen van spelregels. Wie de achterliggende spelregels van een game kan identificeren en hun werkwijze begrijpt, kan deze spelregels immers gericht en effectiever aanpassen en verbeteren. Het tweede ontwerp-kader kan game designers helpen bij het creëren van een open en tegelijkertijd dynamische wereld die spelers meezuigt in een soepele spelervaring. De twee ontwerptheorieën staan overigens niet los van elkaar: in een goede game vormen de spelwereld en de spelregels een complexe, goed uitgebalanceerde eenheid. Het een kan niet zonder het ander en andersom.

Het eerste kader dat game designers inzicht biedt in spelregels en hun werking heet *Machinations* en maakt gebruik van dynamische, interactieve diagrammen. Deze diagrammen brengen de interne economie van een game in kaart door de stroom van belangrijke, en soms abstracte, resources te modelleren. Resources kunnen grondstoffen zijn. Maar ook punten, geld of kogels.

De belangrijkste structurele eigenschap die *Machinations*-diagrammen voor het voetlicht brengen zijn feedbackpatronen. Feedback wordt over het algemeen gezien als een belangrijke oorzaak van dynamische gedrag in complexe systemen zoals games. Feedback zorgt ervoor dat een statusverandering van één game-element andere game-elementen beïnvloedt waardoor vervolgens het eerste element weer verandert. In games heeft dit vaak een zelfversterkend effect. Denk aan het klassieke bordspel *Monopoly*. Wie daar geld investeert in vastgoed verdient dat uiteindelijk ruimschoots terug door huuropbrengsten en kan daardoor nog meer investeren.

Het *Machinations*-kader maakt onderscheid tussen verschillende soorten feedback: feedback kan positief of negatief zijn, constructief of destructief, snel of langzaam, etcetera. Voor game designers is het van belang goed zicht te hebben op de feedbackwerking in hun games en deze te bestuderen en waar nodig aan te passen. Zij kunnen daarbij gebruik maken van patronen in interne spel-economieën die steeds weer terugkeren. Deze patronen zijn een weerslag van

veel voorkomende oplossingen voor frequente ontwerpproblemen. Machinations diagrammen zijn een uitstekend vehikel om deze patronen te identificeren en ze verder te bestuderen.

Game designers kunnen Machinations-diagrammen maken met de software die voor dit onderzoek is ontwikkeld. Het grootste voordeel van deze software is dat digitale diagrammen net zo dynamisch en interactief zijn als de games die ze representeren. Dit stelt game designers in staat games gemakkelijk en efficiënt te simuleren. Ze kunnen de interne speleconomie modelleren en balanceren nog voordat er een prototype van de game is gemaakt.

Het tweede theoretische kader van dit proefschrift, Mission/Space, richt zich op level-ontwerp en spelmechanismen die de voortgang van een speler bepalen. In tegenstelling tot bestaande modellen voor level-ontwerp, bouwt Mission/Space voort op het idee dat er in een level twee verschillende structuren bestaan. Mission-diagrammen worden gebruikt om de structuur van taken en uitdagingen voor de speler te formaliseren, terwijl space-diagrammen de ruimtelijke constructie formaliseren. Beide constructies zijn aan elkaar gerelateerd, maar zijn niet hetzelfde. De verschillende wijzen waarop missies geprojecteerd kunnen worden op een bepaalde ruimte speelt uiteindelijk een belangrijke rol in de totstandkoming van de spelervaring. Spelmechanismen voor sloten en sleutels, die vaak gebruikt worden om te voorkomen dat spelers te snel een bepaald doel bereiken, spelen hier bij een belangrijke rol.

Zoals in het geval van Machinations leent het Mission/Space-kader zich voor het ontwikkelen van software die ontwikkelaars kan ondersteunen in het ontwerpen van game levels. Het programma Ludoscope implementeert mission- en space-diagrammen en stelt ontwikkelaars op die manier in staat om de voortgang van spelers door een level te simuleren. Zo helpt Ludoscope in een vroeg stadium met het identificeren van potentiële fouten in het ontwerp.

Waar Mission/Space zich richt op level-ontwerp en games of progression, richt Machinations zich meer op interne speleconomie en games of emergence. Door beide kaders te combineren ontstaat een nieuw perspectief waarmee de combinatie van emergence en progression kan worden onderzocht. Veel games combineren beide structuren. Toch is er nog een groot aantal spellen waar de rijkdom van de mechanismen van de speleconomie, ruimte en fysieke simulatie niet aansluiten bij de beperkte mechanismen die de spelerprogressie controleren. Door Machinations-diagrammen te combineren met mission- en space-diagrammen, worden twee nieuwe strategieën inzichtelijk waarmee game designers emergence en progression verder kunnen integreren.

Een eerste combinatiestrategie richt zich op slot- en sleutelmechanismen die eigenlijk zelden tot nooit gebruik maken van feedbackmechanismen. Met Machinations-diagrammen kunnen designers echter op eenvoudige wijze meer geraffineerde en emergente spelmechanismen ontwikkelen voor sloten en sleutels. Een tweede combinatiestrategie ontstaat door de voortgang van spelers als resource te modelleren waardoor het mogelijk wordt deze te betrekken in feedbackmechanismen op de schaal van het complete level of spel.

De Machinations- en Mission/Space-kaders formaliseren verschillende perspectieven op games. 'Model driven engineering', een begrip ontleend aan soft-

ware engineering, kan beide perspectieven verenigen onder een formele benadering van game ontwikkeling. Wie games op een goede manier formeel weet te benaderen, kan uiteindelijk bepaalde onderdelen van het ontwikkeltraject automatiseren. Door gebruik te maken van formele grammatica's en herschrijfsystemen is het bijvoorbeeld mogelijk om transformaties vast te leggen die game designers kunnen gebruiken om ruimtes te genereren voor een missie, spelmechanismen voor een ruimte, of een missie voor spelmechanismen. Door het ontwikkelproces onder te verdelen in een serie kleine transformaties wordt het mogelijk flexibele software tools te maken die game levels en, tot op zekere hoogte, zelfs hele games automatisch kunnen (helpen) genereren. Een andere toepassing is het ontwikkelen van games die zich automatisch aanpassen aan de prestaties van spelers.

Om het automatiseren van game design dichterbij te brengen is voor dit proefschrift het prototype Ludoscope verder uitgebouwd. Het stelt designers in staat grammatica's en herschrijfsystemen zelf te definiëren. In potentie maakt dit Ludoscope tot een zeer krachtige, generieke game ontwikkeltool die het mogelijk maakt om het ontwerpproces op een hoger abstractieniveau te brengen.

De software, ontwikkeld als onderdeel van dit onderzoek, is tegelijkertijd ook een manier om de theorie uit dit proefschrift uit te proberen en te valideren. Maar het is niet de enige wijze van valideren die is toegepast. Het overgrote deel van de ideeën uit dit proefschrift zijn gepresenteerd, uitgetoetst en besproken op lezingen en in workshops. Academici en ervaren game designers hebben zo de mogelijkheid gekregen commentaar te leveren en het werk te beïnvloeden. Daar komt bij dat beide theoretische kaders, maar vooral *Machinations*, gebruikt zijn in het game development onderwijs aan de Hogeschool van Amsterdam. Daaruit is gebleken dat de kaders studenten helpen om games beter te begrijpen en te verbeteren. Vooral doordat ze hen een formeel perspectief bieden op belangrijke aspecten van game ontwikkeling die zonder de theoretische kaders niet meteen zicht- of tastbaar zijn. De theoretische kaders stelden de studenten in staat onderdelen van games te simuleren en er mee te experimenteren.

Het ontwikkelen van games blijft een uitdaging. De theorie en theoretische gereedschappen die in dit proefschrift zijn gepresenteerd kunnen game ontwerpers helpen doordat ze domeinkennis vastleggen, ontwerpers in staat stellen games in een vroeg stadium te simuleren, en zelfs door gedeeltes van het ontwerpproces te automatiseren. Maar ze kunnen natuurlijk nooit de menselijke creativiteit en vindingrijkheid vervangen die noodzakelijke ingrediënten blijven voor het maken van games. Uiteindelijk denk ik dat deze tools en theorie ontwerpers kunnen helpen en een bijdrage kunnen leveren om game design naar een hoger niveau te tillen.

Index

- 400 project, 50
- Aarseth, Espen, 2, 135
- Adams, Ernest, 4, 6, 7, 44, 48, 60, 71, 80, 81, 97, 110, 114, 187, 209
- affordance, 27
- agency, 115
- Agricola, 142
- Alexander, Christopher, 34, 52, 53
- America's Army, 29
- Angry Birds, 8, 9, 15
- Araújo, Manual, 56
- Arinbjarnar, Maria, 114
- Arneson, Dave, 130
- Ascent, 206
- Ashmore, C, 128
- Assassins Creed, 12
- augmented transition networks, 55

- Backgammon, 219
- Bahktin, Mikhail M., 116
- Bakkes, Sander, 176, 193, 207
- Baldur's Gate, 12
- Ball, Philip, 16
- Barwood, Hal, 50
- Beck, John, 33
- Bejeweled, 15
- Bewbees, 206
- Binmore, Ken, 57
- Björk, Staffan, 51–53, 180
- Blackjack, 58, 59
- Blood Bowl, 250, 251
- Bogost, Ian, 25, 27, 29
- Boulder Dash, 8, 189
- Brom, Cyril, 56
- Brown, Alan, 162
- Bura, Stéphane, 58, 208
- Burke, Edmund, 28
- Burke, Timothy, 71
- Byrne, Ed, 109, 110, 113

- Caesar III, 142–146, 158, 159, 231
- Campagne, 206
- Campbell, Joseph, 129
- Castronova, Edward, 82
- Caylus, 77, 78, 142, 146, 251
- cellular automata, 16, 186, 210

- Chalmers, David J., 19
- Checkers, 58
- Chess, 12, 13, 16, 34, 59, 84, 88, 107, 147
- Chomsky, Noam, 34, 55, 164, 165
- chronotope, 116
- Church, Doug, 21, 49
- Civilization, 15, 21, 142–144, 147, 233, 234
- Civilization III, 15, 244, 251
- Colossal Cave Adventure, 15
- Command & Conquer: Red Alert, 12
- commodity flow graphs, 61
- complexity, 16
- Compton, Kate, 125
- Connect Four, 35, 36
- control theory, 83, 84
- Conway, John, 19
- Cook, Daniel, 51, 138
- Copier, Marinka, 5
- Costikyan, Greg, 16
- Counter-Strike, 14
- Cousins, Ben, 113
- Crawford, Chris, 16, 26, 37, 190
- Cubel, José Á Carsí, 162

- data intensity, 37
- Descent: Journeys in the Dark, 218, 219
- design patterns, 51–53, 91
 - attrition, 234–238
 - converter engine, 223–226
 - dynamic engine, 92, 101, 152, 153, 157, 220–223
 - dynamic friction, 153, 231–234
 - engine building, 92, 227–229
 - escalating complexity, 156, 246–248
 - escalating complications, 155, 156, 158, 244–246
 - multiple feedback, 240–242
 - playing style reinforcement, 248–251
 - static engine, 217–220
 - static friction, 157, 229–231
 - stopping mechanism, 238–240
 - trade, 242–244

- Deus Ex, 14, 30, 114, 147–149, 190
- Diablo, 30, 31, 37, 186
- discrete infinity, 34
- DiStefano III, Joseph J., 83
- Djaouti, Damien, 180
- Donkey Kong, 8
- Dormans, Joris, 27, 59, 67, 148, 176, 187, 191, 193, 207
- Doull, Andrew, 161
- Dungeon Run, 206
- Dungeons & Dragons, 130, 186, 187
- Elite, 226
- emergence, 13–20, 23, 35, 36, 54, 68, 142, 144
- Enchanted Hands, 205
- Eskelinen, Markku, 2
- EverQuest, 14
- Eye of the Beholder II, 125
- Fable, 190
- Fallout 3, 16, 82
- Falstein, Noah, 50
- Fearsome Floors, 219
- feedback, 18, 23, 60, 83, 151, 152, 155, 198
 - determinability, 88–90
 - multiple feedback loops, 84
 - negative, 84
 - patterns, 91
 - positive, 84
 - profiles, 87–91
- Flix, 206
- Flix 2, 206
- formal grammars, 164–166
 - classification, 165
 - context-free grammars, 166
 - context-sensitive grammars, 165
 - graph grammars, 168
 - nonterminals, 164
 - regular grammars, 166
 - rewrite rules, 165
 - shape grammars, 170
 - terminals, 164
 - unrestricted grammars, 165, 169
- Fowler, Martin, 54
- Frasca, Gonzalo, 3, 25
- Friedman, Ted, 21
- Fromm, Jochen, 18
- Fron, Janine, 84
- Fullerton, Tracy, 4, 48, 83, 109
- Gabbler, Kyle, 136
- Galloway, Alexander R., 4
 - game
 - definition, 5, 22
 - genre, 10–13
 - game design documents, 44–46
 - one-page, 45
 - game diagrams, 57–59
 - Game of Goose, 219
 - Game of Life, 19
 - game ontology project, the, 50
 - game studies, 2
 - game vocabulary, 49–51, 62
 - gameplay, 5, 23, 35, 68, 198
 - adaptable, 188
- Gamma, Erich, 52
- Genette, Gérard, 115
- Get H2O, 38, 39, 206
- Gips, James, 170
- Go, 12, 16, 34, 107, 210
- Grünvogel, Stefan M., 53
- Grand Theft Auto III, 15
- Grand Theft Auto: San Andreas, 14
- graph grammars, 168–170
 - deletion, 169
- Guardiola, Emmanuel, 57
- Guttenberg, Darren, 62
- Half-Life, 12, 21, 130
- Half-Life 2, 20, 21, 130, 148
- Halo, 12
- Heckel, Reiko, 168
- hero's journey, 116
- hierarchy of challenges, 114
- Holland, John H., 67
- Holopainen, Jussi, 51–53, 180
- Hooft, Niels 't, 5
- Hopson, John, 97
- Huizinga, Johan, 5, 40
- human-centered design, 48

- Hunicke, Robin, 47
- hyperrealism, 30
- in medias res, 129
- Infinite Boulder Dash, 189, 190, 206
- Infinite Mario Bros., 189, 192
- innovation, 10, 51
- internal economy, 7, 23, 59, 71, 83, 198
- Järvinen, Aki, 53
- Jakobson, Roman, 28
- Jenkins, Henry, 20, 110
- Jenkins, Odest Chadwicke, 6, 61, 71
- Johnson, Lawrence, 186
- Johnson, Steven, 48
- Juul, Jesper, 1, 2, 4, 13, 16, 26, 36, 54, 144, 188, 210
- Keith, Clinton, 45
- Kim, K. L., 27
- King, Geoff, 30
- Klabbers, Jan, 36
- Klevjer, Rune, 25
- Klop, Jan Willem, 166, 167
- Knytt Stories, 139, 140, 142
- Kohler, Chris, 138
- Koster, Raph, 3, 57, 61
- Kreimeier, Bernd, 45, 49, 52
- Kriegsspiel, 32, 37, 97
- Lakoff, George, 38
- language, 28, 34
- Laurel, Brenda, 2
- LeBlanc, Marc, 47, 48, 83, 84, 86, 88
- level, 23
- level design, 109, 199
 - boss character, 33, 125, 190
 - bottleneck, 128
 - branching, 110
 - hub-and-spoke, 111, 128, 130
 - kata, 138
 - kihon, 138
 - learning curve, 20, 137, 182, 187
 - mini-boss, 125, 130
 - pacing, 187
 - railroading, 21, 110, 130
 - tasks, 113
 - tutorials, 20
- Lewis, Chris, 210
- Librande, Stone, 45
- Lindley, Craig, 50
- Lister, Martin, 37
- Little Big Planet, 192
- LKE, 206–208
- Locke, John, 28
- ludology, 2, 50
- Ludoscope, 131, 193–196, 205, 211
- Machinations
 - activation modes, 72, 73, 94
 - activators, 77
 - artificial players, 96
 - charts, 94
 - connection label, 73
 - converters, 81, 82
 - drains, 81
 - end conditions, 94
 - framework, 68–71
 - game state, 72, 75
 - gates, 78–80
 - inputs, 73
 - label modifiers, 75
 - modifiers
 - intervals, 100
 - random, 97
 - node modifiers, 76
 - outputs, 73
 - conditional, 78
 - probable, 78
 - pools, 73
 - pull modes, 94
 - randomness, 97
 - resource connections, 73
 - resources, 71
 - color-coded, 96
 - sources, 81
 - state connections, 75–78
 - static diagrams, 72
 - time modes, 72, 74
 - traders, 82
 - triggers, 77
- magic circle, 4, 5, 20
- Magic the Gathering, 236, 237

- Malaby, Thomas M., 70
 Martin, Paul, 131
 Mass Murderer Game, 3
 Mateas, Michael, 125, 180, 187
 McCloud, Scott, 109
 McGuire, Morgan, 6, 61, 71
 MDA framework, 46–48, 136, 201
 mechanics, 6–10, 23, 46, 68
 and genre, 12
 continuous versus discrete, 7–9
 controlling progression, 12, 146, 147, 149, 154–159
 core, 6, 23
 dialog trees, 148
 dice, 32
 double jumping, 139
 economy building, 142
 inventory, 30
 jumping, 33
 locks and keys, 128, 130, 139, 142, 149, 152–154, 174–175, 199
 physics, 7, 209
 power-ups, 139
 social interaction, 12, 210
 tactical maneuvering, 12, 210
 types, 12
 MineCraft, 186
 Mission/Space
 entrances, 118
 force-ahead, 125
 framework, 115–117
 game element, 122
 goals, 118
 hub, 124
 inhibition, 117
 lock, 122
 lock relation, 123
 locked short-cut, 124
 mission graphs, 117–121, 131
 mission state, 117, 119
 task state, 117–118
 tasks, 117
 mission-space morphology, 129
 open space, 124
 path, 122
 pathway, 124
 place, 121
 requirements, 119–121
 strong, 117
 weak, 117
 set-back, 124
 space graphs, 121–125, 131
 tasks, 118
 unlock relation, 123
 valve, 123
 window, 123
 Miyamoto, Shigeru, 187
 model driven engineering, 162
 model transformation, 162–164, 187, 192
 monomyth, 129
 Monopoly, 71, 73, 75–77, 81–84, 92, 153, 221, 231
 Murray, Janet, 2, 28, 115
 Myers, David, 28, 36
 narratology, 2, 50
 Natkin, Stéphane, 56, 57
 Nelson, Mark J., 180, 187
 Nietzsche, M., 128
 Nintendo, 116, 138
 Pac-Man, 28, 155, 246
 Pagulayan, Randy J., 48
 Panofsky, Erwin, 63
 Peirce, Charles S., 27
 Petri nets, 56, 57
 play-centric design, 48–49, 201
 player modeling, 190
 plot, 115
 Poole, Steven, 26, 28, 40
 Pooley, Rob, 59
 Portal, 9
 Power Grid, 60, 61, 88, 225, 226, 240
 Prince of Persia, 123
 Prince of Persia: Sands of Time, 130
 procedural content, 186–188
 mixed-initiative approach, 192, 194
 process intensity, 37
 progression, 12–15, 20, 23, 36, 54, 142
 Propp, Vladimir, 129
 Puerto Rico, 142, 228
 quantifiable outcome, 4, 5

- quests, 147
- realism, 27
- recursion, 55
- Rekers, J., 168
- representation, 27
- rewrite system
 - confluency, 167
 - normal form, 167
- rewrite systems, 166–168, 187, 192
 - generating learning curves, 182
 - locks and keys, 174
 - Machinations, 180
 - to generate space, 176
- Reyno, Emanuel Montero, 162
- Risk, 14, 31, 32, 55, 56, 71, 78, 81, 84–88, 91, 92, 97, 242
- Robot Wants Kitty, 139
- Rogers, Scott, 44
- Rogue, 186
- roguelike games, 186–187
- Rollings, Andrew, 4, 6, 7, 44, 48, 60, 71, 80, 81, 97, 110, 114, 187
- Rosenblum, Robert, 63
- rules, 4, 6, 10
- Rumbaugh, Jim, 59
- Ryan, Marie-Laure, 2, 40, 110, 191
- Saint-Exupéry, Antoine de, 25, 34
- Salen, Katie, 4, 14, 48, 57, 83, 84
- sandbox games, 15
- Saussure, Ferdinand de, 28
- Schürr, A., 168
- Schaffer, Noah, 48
- Schell, Jessie, 110, 176
- Seasons, 156–159, 185, 206
- Selic, Bran, 59, 162
- semiotics, 27
- September 12, 3
- Settlers of Catan, 62, 76, 82, 91, 92, 97, 222, 223, 229, 244
- Shannon, Claude E., 34
- shape grammars, 170–172
 - line-segments, 171
 - points, 171
 - quads, 171
- Sheffield, Brandon, 62
- signs, 27
- SimCity, 13–16, 100, 159, 228
- Simpson, Zack, 71
- simulation, 25, 27, 29, 50
 - iconic, 28, 33
 - indexical, 30–31, 33, 37, 39
 - symbolic, 31–33, 37, 39
- SimWar, 68, 96, 100–103, 105, 236
- Smelik, Ruben, 192
- Smith, Gillian, 187, 192
- Smith, Harvey, 16, 36, 37
- Snakes and Ladders, 219
- software engineering, 48, 52, 162
- Space Hulk, 237, 238
- Space Invaders, 155, 245
- Spector, Warren, 30
- Spore, 136, 137, 186
- SPUG, 137
- Star Defender, 119, 120
- Star Wars: X-Wing Alliance, 81, 218, 219
- StarCraft, 12, 13, 92, 93, 142, 148, 222
- StarCraft II, 147, 149
- state machines, 53, 54
- Stephenson, Neal, 191
- Stevens, Perdita, 59
- Stiny, George, 170
- story, 147, 190
- Super Mario Bros., 7, 8, 27, 32, 33, 37, 38, 121, 189
- Super Mario Kart, 84
- Super Mario Sunshine, 113, 114
- Swain, Chris, 48
- Sweetser, Penelope, 36, 48
- System Shock 2, 115, 164
- Tangram, 15
- Taylor, M. J., 59
- Tetris, 3, 28, 88, 156, 248
- The Elder Scrolls IV: Oblivion, 12, 16, 251
- The Game of Goose, 154
- The Landlord Game, 84
- The Legend of Zelda, 12, 40, 125, 174, 188, 253

- The Legend of Zelda: Twilight Princess,
40, 110, 111, 116, 125–127, 133,
138, 139, 141, 142, 148, 151
- The Sims, 100
- Tic-Tac-Toe, 35, 59
- Togelius, Julian, 188, 189
- Tolkien, J.R.R., 20
- Torchlight, 186
- Tower of Goo, 9
- Unified Modeling Language, 54, 59, 162
- Up the River, 219
- Veugen, Connie, 10
- Vogler, Christopher, 63, 116, 129
- Wade, Mitchell, 33
- Ward, J., 26
- Wardrip-Fruin, Noah, 27, 147–149, 190
- Warhammer Fantasy Roleplay, 154, 155
- Weyth, Peta, 48
- Wolf, Mark J. P., 4
- Wolfram, Stephen, 16–18
- Woods, William A., 55
- World of Goo, 9
- Zagal, José, 51
- Zimmerman, Eric, 4, 14, 48, 57, 83, 84
- Zork, 15

Bibliography

- Aarseth, E. J. (2001). Computer Game Studies, Year One. *Game Studies*, 1(1).
URL <http://www.gamestudies.org/0101/editorial.html>
- Aarseth, E. J. (2005). From Hunt the Wumpus to EverQuest: Introduction to Quest Theory. In *Proceedings of the International Conference on Entertainment Computing, Sanda Japan, September 2005*, (pp. 496–506).
- Adams, E. (2000). A Letter From A Dungeon. *Gamasutra*.
URL http://www.gamasutra.com/view/feature/3424/the_designers_notebook_a_letter_.php
- Adams, E., & Rollings, A. (2007). *Fundamentals of Game Design*. Upper Saddle River, NJ: Pearson Education, Inc.
- Alexander, C. (1979). *The Timeless Way of Building*. New York, NY: Oxford University Press.
- Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language: Towns - Buildings - Construction*. New York, NY: Oxford University Press.
- Araújo, M., & Roque, L. (2009). Modeling Games with Petri Nets. In *Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference, London UK, September 2009*.
- Arinbjarnar, M., & Kudenko, D. (2009). Directed Emergent Drama vs. Pen & Paper Role-Playing Games. In *Proceedings of the AISB'09 Symposium: AI & Games, Edinburgh UK, April 2009*.
- Arvo, J., & Novins, K. (2007). Stratified Sampling of Convex Quadrilaterals.
URL <http://www.ics.uci.edu/~arvo/EECS204/papers/quad.pdf>
- Ashmore, C., & Nietsche, M. (2007). The Quest in a Generated World. In *Situated Play: Proceedings of the 2007 Digital Games Research Association Conference, Tokyo Japan, September 2007*, (pp. 503–509).
- Bahktin, M. M. (1981). *The Dialogic Imagination*. Austin, TX: University of Texas Press.
- Bakkes, S., & Dormans, J. (2010). Generating Mission and Space for Dynamic Play Experience. In *Proceedings of the GAME-ON Conference, Leicester UK, November 2010*, (pp. 72–79).
- Ball, P. (2004). *Critical Mass: How One Thing Leads To Another*. New York, NY: Farrar, Straus and Giroux.
- Beck, J. C., & Wade, M. (2004). *Got Game: How the Gamer Generation Is Reshaping Business Forever*. Boston, MA: Harvard Business School Press.

- Binmore, K. (2007). *Game Theory: A Very Short Introduction*. Oxford, UK: Oxford University Press.
- Björk, S., & Holopainen, J. (2005). *Patterns in Game Design*. Boston, MA: Charles River Media.
- Bogost, I. (2006). *Unit Operations: An Approach to Videogame Criticism*. Cambridge, MA: The MIT Press.
- Bogost, I. (2007). *Persuasive Games: The Expressive Power of Videogames*. Cambridge, MA: The MIT Press.
- Brathwaite, B. (2010). My games: the mechanic is the message. Private blogpost.
URL <http://bbrathwaite.wordpress.com/2010/03/14/my-games-the-mechanic-is-the-message/>
- Brom, C., & Abonyi, A. (2006). Petri-Nets for Game Plot. In *Proceedings of AISB '06: Adaptation in Artificial and Biological Systems, Bristol UK, April 2006*, (pp. 6–13).
- Brown, A. (2004). An introduction to Model Driven Architecture.
URL <http://www.ibm.com/developerworks/rational/library/3100.html>
- Bura, S. (2006). A Game Grammar.
URL <http://www.stephanebura.com/diagrams/>
- Burke, E. (1990). *A Philosophical Enquiry into the Origin of our Ideas of the Sublime and Beautiful*. Oxford, UK: Oxford University Press.
- Burke, T. (2005). Can A Table Stand On One Leg? Critical and Ludological Thoughts on Star Wars: Galaxies. *Game Studies*, 5(1).
URL <http://www.gamestudies.org/0501/burke/>
- Byrne, E. (2005). *Game Level Design*. Boston, MA: Charles River Media.
- Campbell, J. (1947). *The Hero With A Thousand Faces*. Princeton, NJ: Princeton University Press.
- Castronova, E. (2005). *Synthetic Worlds: The Business and Culture of Online Games*. Chicago, IL: The University of Chicago Press.
- Chalmers, D. J. (2006). Strong and Weak Emergence. In P. Clayton, & P. Davies (Eds.) *The Re-emergence of Emergence*, (pp. 244–256). Oxford, UK: Oxford University Press.
- Chomsky, N. (1957). *Syntactic Structures*. The Hague, the Netherlands: Mouton Publishers.
- Chomsky, N. (1959). On Certain Formal Properties of Grammars. *Information and Control*, 2(2), 137–167.

- Chomsky, N. (1972). *Language and Mind, Enlarged Edition*. New York, NY: Harcourt Brace Jovanovich Inc.
- Church, D. (1999). Formal Abstract Design Tools. *Gamasutra*.
URL http://www.gamasutra.com/features/19990716/design_tools_01.htm
- Compton, K., & Mateas, M. (2006). Procedural Level Design for Platform Games. In *Proceedings of AAAI Conference, Boston MA, July 2006*, (pp. 109–111).
- Cook, D. (2006). GameInnovation.org. Blog post on LostGarden.
URL <http://www.lostgarden.com/2006/04/gameinnovationorg.html>
- Cook, D. (2007). The Chemistry Of Game Design. *Gamasutra*.
URL http://www.gamasutra.com/view/feature/1524/the_chemistry_of_game_design.php
- Copier, M. (2007). *Beyond the Magic Circle: A Network Perspective on Role-Play in Online Games*. Ph.D. thesis, University Utrecht, The Netherlands.
- Costikyan, G. (1994). I Have No Words I Must Design. *Interactive Fantasy*.
URL <http://www.interactivedramas.info/papers/nowordscostikyan.pdf>
- Cousins, B. (2004). Elementary game design. *Develop*, (pp. 51–54).
- Crawford, C. (1984). *The Art of Computer Game Design*. Berkeley, CA: McGraw Hill/Osborne Media.
URL <http://vancouver.wsu.edu/fac/Peabody/game-book/coverpage.html>
- Crawford, C. (2003a). *Chris Crawford on Game Design*. Indianapolis, Indiana: New Riders Publishing.
- Crawford, C. (2003b). Interactive Storytelling. In M. J. P. Wolf, & B. Perron (Eds.) *The Video Game Theory Reader*, (pp. 259–273). New York, NY: Routledge.
- Crawford, C. (2005). *Chris Crawford on Interactive Storytelling*. Berkeley, CA: New Riders Publishing.
- DeMaria, R., & Wilson, J. L. (2004). *High Score! The illustrated history of electronic games*. Emeryville, CA: McGraw-Hill/Osborne.
- DiStefano III, J. J., Stubberud, A. R., & Williams, I. J. (1967). *Theory and Problems of Feedback and Control Systems*. New York, NY: McGraw-Hill.
- Djaouti, D., Alvarez, J., Jessel, J.-P., Methel, G., & Molinier, P. (2008). A Gameplay Definition through Videogame Classification. *International Journal of Computer Games Technology*.

- Dormans, J. (2005). The Art of Jumping.
URL <http://www.jorisdormans.nl/article.php?ref=artofjumping>
- Dormans, J. (2006a). Lost in a Forest: Finding New Paths for Narrative Gaming. Article posted on Game-Research.org.
URL <http://game-research.com/index.php/articles/lost-in-a-forest-finding-new-paths-for-narrative-gaming/>
- Dormans, J. (2006b). On the Role of the Die: A brief ludologic study of pen-and-paper roleplaying games and their rules. *Game Studies*, 6(1).
URL <http://gamestudies.org/0601/articles/dormans>
- Dormans, J. (2006c). Talking to Games: Gameplay as Expressive Performance. In *Proceedings of the International Digital Games Conference, Portalegre Portugal, September 2006*, (pp. 133–143).
- Dormans, J. (2008a). Beyond Iconic Simulation. In *Proceedings of the IADIS Gaming Conference, Amsterdam The Netherlands, July 2008*, (pp. 51–58).
- Dormans, J. (2008b). Visualizing Game Mechanics and Emergent Gameplay. In *Proceedings of the Meaningful Play Conference, East Lansing MI, October 2008*.
- Dormans, J. (2009). Machinations: Elemental Feedback Structures for Game Design. In *Proceedings of the GAMEON-NA Conference, Atlanta GA, August 2009*, (pp. 33–40).
- Dormans, J. (2010). Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. In *Proceedings of the Foundations of Digital Games Conference Monterey CA, June 2010*.
- Dormans, J. (2011a). Beyond Iconic Simulation. *Simulation and Gaming, Special Issue*, 42(5), 610–631.
- Dormans, J. (2011b). Integrating Emergence and Progression. In *Think Design Play: Proceedings of the 2011 Digital Games Research Association Conference, Hilversum the Netherlands, September 2011*.
- Dormans, J. (2011c). Level Design as Model Transformation: A Strategy for Automated Content Generation. In *Proceedings of the Foundations of Digital Games Conference, Bordeaux France, June 2011*.
- Dormans, J. (2011d). Simulating Mechanics to Study Emergence in Games. In *Proceedings of AI and Interactive Digital Entertainment Conference, Palo Alto CA, October 2011*.
- Dormans, J., & Bakkes, S. (2011). Generating Missions and Spaces for Adaptable Play Experiences. *IEEE Transactions on Computational Intelligence and AI in Games - Special Issue on Procedural Content Generation*, 3(3), 216–228.

- Doull, A. (2008). The Death of the Level Designer.
URL <http://pcg.wikidot.com/the-death-of-the-level-designer>
- Eskelinen, M. (2001). Towards Computer Game Studies. In *Proceedings of SIGGRAPH 2001, Art Gallery, Art and Culture Papers*, (pp. 83–87).
URL <http://www.siggraph.org/artdesign/gallery/S01/essays/0416.pdf>
- Falstein, N. (2002). Better By Design: The 400 Project. *Game Developer Magazine*, 9(3), 26.
- Foster, E. M. (1962). *Aspects of the Novel*. Harmondsworth, UK: Penguin Books Ltd.
- Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Boston, MA: Addison-Wesley, third ed.
- Frasca, G. (1999). Ludology Meets Narratology: Similitude and differences between (video)games and narrative.
URL <http://www.ludology.org/articles/ludology.htm>
- Frasca, G. (2003). Simulation versus Narrative. In M. J. P. Wolf, & B. Perron (Eds.) *The Video Game Theory Reader*, (pp. 221–235). New York, NY: Routledge.
- Friedman, T. (1999). Civilization and Its Discontents: Simulation, Subjectivity, and Space. In G. Smith (Ed.) *On a Silver Platter: CD-ROMs and the Promises of a New Technology*, (pp. 132–150). New York, NY: New York University Press.
- Fromm, J. (2005). Types and Forms of Emergence.
URL <http://arxiv.org/abs/nlin.AO/0506028>
- Fron, J., Fullerton, T., Morie, J. F., & Pearce, C. (2007). The Hegemony of Play. In *Situated Play: Proceedings of the 2007 Digital Games Research Association Conference, Tokyo Japan, September 2007*, (pp. 309–318).
- Fullerton, T. (2008). *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. Burlington, MA: Morgan Kaufman, 2nd ed.
- Fullerton, T., Chen, J., Santiago, K., Nelson, E., Diamante, V., Meyers, A., Song, G., & DeWeese, J. (2006). That Cloud Game: Dreaming (and Doing) Innovative Game Design. In *Sandbox '06 Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, (pp. 51–59).
- Gabler, K. (2009). How to make a game in 48 hours. Video keynote for the Global Game Jam 2009.
URL <http://www.youtube.com/watch?v=aW6vgW8wc6c>
- Galloway, A. R. (2006). *Gaming: Essays on Algorithmic Culture*. Minneapolis, MN: University of Minnesota Press.

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison Wesley.
- Gardner, M. (1970). Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, (223), 120–123.
URL http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_gamelife/ConwayScientificAmerican.htm
- Genette, G. (1980). *Narrative Discourse*. Oxford, UK: Basil Blackwell.
- Grünvogel, S. M. (2005). Formal Models and Game Design. *Game Studies*, 5(1).
URL <http://gamestudies.org/0501/gruenvogel/>
- Guardiola, E., & Natkin, S. (2005). Game Theory and Video Game, A New Approach of Game Theory to Analyze and Conceive Game Systems. In *Proceedings of the 7th International Conference on Computer Games, Angoulême France, November 2005*.
- Guttenberg, D. (2006). An Academic Approach to Game Design: Is It Worth It? *Gamasutra*.
URL http://www.gamasutra.com/view/news/8908/Student_Feature_An_Academic_Approach_to_Game_Design_Is_It_Worth_It.php
- Heckel, R. (2006). Graph Transformation in a Nutshell. *Electronic Notes in Theoretical Computer Science* 148, (pp. 187–198).
- Hodgson, D. S., & Stratton, S. (2006). *The Legend of Zelda: Twilight Princess: PRIMA Official Game Guide*. Roseville, CA: Prima Games.
- Holland, J. H. (1998). *Emergence: From Chaos to Order*. Oxford, UK: Oxford University Press.
- Hopson, J. (2001). Behavioral Game Design. *Gamasutra*.
URL http://www.gamasutra.com/view/feature/3085/behavioral_game_design.php
- Huizinga, J. (1997). *Homo Ludens: Proeve Ener Bepaling Van Het Spelelement Der Cultuur*. Amsterdam, The Netherlands: Pandora.
- Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA: A formal approach to game design and game research. In *Proceedings of the AAAI-04 Workshop on Challenges*, (pp. 1–5).
- Jakobson, R. (1960). Closing Statement: Linguistics and Poetics. In T. A. Sebeok (Ed.) *Style in Language*, (pp. 350–378). Cambridge, MA: The MIT Press.
- Järvinen, A. (2003). Making and Breaking Games: A Typology of Rules. In M. Copier, & J. Raessens (Eds.) *Level Up Conference Proceedings: Proceedings of the 2003 Digital Games Research Association Conference, Utrecht The Netherlands, November 2003*, (pp. 68–79).

- Jenkins, H. (2004). Game Design as Narrative Architecture. In N. Wardrip-Fruin, & P. Harrigan (Eds.) *First Person: New Media as Story, Performance and Game*, (pp. 118–130). Cambridge, MA: The MIT Press.
- Johnson, L., Yannakakis, G. N., & Togelius, J. (2010). Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the Foundations of Digital Games Conference 2010, Monterey, CA*.
- Johnson, S. (2005). *Everything Bad is Good for You, How Popular Culture is Making Us Smarter*. London, UK: Penguin Boos Ltd.
- Juul, J. (2002). The Open and the Closed: Games of Emergence and Games of Progression. In F. Mäyrä (Ed.) *Proceedings of Computer Games and Digital Cultures Conference, Tampere Finland, June 2002*, (pp. 323–329).
- Juul, J. (2003). Just what is it that makes computer games so different, so appealing? Ivory Tower Column for IGDA.
- Juul, J. (2005). *Half-Real: Video Games between Real Rules and Fictional Worlds*. Cambridge, MA: The MIT Press.
- Juul, J. (2009). *A Casual Revolution: Reinventing Video Games and Their Players*. Cambridge, MA: The MIT Press.
- Juul, J. (2010). At What Cost Failure? When Games Punish Players. In *Proceedings of the Foundations of Digital Games Conference, Monterey CA, June 2010*, (pp. 86–91).
- Keith, C. (2010). *Agile Game Development with Scrum*. Upper Saddle River, NJ: Addison-Wesley.
- Kim, K. L. (1996). *Caged in Our Own Signs: A Book About Semiotics*. Norwood, NJ: Ablex Publishing Corporation.
- King, G. (2000). *Spectacular Narratives, Hollywood in the Age of the Blockbuster*. London, UK: I.B. Tauris Publishers.
- Klabbers, J. H. G. (2006). *The Magic Circle: Principles of Gaming & Simulation*. Rotterdam, The Netherlands: Sense Publishers.
- Klevjer, R. (2002). In Defense of Cutscenes. In F. Mäyrä (Ed.) *Proceedings of Computer Games and Digital Cultures Conference, Tampere Finland, June 2002*, (pp. 191–202).
- Klop, J. W. (1992). Term Rewriting Systems. In S. Abramsky, D. Gabbay, & M. T. (Eds.) *Handbook of Logic in Computer Science, Volum2. Background: Computational Structures*, (pp. 1–116). Oxford, UK: Oxford University Press.
- Kohler, C. (2005). *Power Up: How Japanese Video Games Gave the World an Extra Life*. Indianapolis, IN: Brady Games.

- Koster, R. (2005a). A Grammar of Gameplay: game atoms: can games be diagrammed? Presentation at the Game Developers Conference, San Francisco CA, March 2005.
URL <http://www.raphkoster.com/gaming/atof/grammarofgameplay.pdf>
- Koster, R. (2005b). *A Theory of Fun for Game Design*. Scottsdale, AZ: Paraglyph Press, Inc.
- Kreimeier, B. (2002). The Case For Game Design Patterns. *Gamasutra*.
URL http://www.gamasutra.com/view/feature/4261/the_case_for_game_design_patterns.php
- Kreimeier, B. (2003). Game Design Methods: A 2003 Survey. *Gamasutra*.
URL http://www.gamasutra.com/view/feature/2892/game_design_methods_a_2003_survey.php
- Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. Chicago, IL: The University of Chicago Press.
- Laurel, B. K. (1986). *Towards the Design of a Computer-Based Interactive Fantasy System*. Ph.D. thesis, Ohio State University.
- LeBlanc, M. (1999). Formal Design Tools: Feedback Systems and the Dramatic Structure of Completion. Presentation at the Game Developers Conference, San Jose CA, March 1999.
URL <http://algorithmancy.8kindsoffun.com/cgdc99.ppt>
- LeBlanc, M. (2004). Mechanics, Dynamics, Aesthetics: A Formal Approach to Game Design. Lecture at Northwestern University.
URL <http://algorithmancy.8kindsoffun.com/MDAnwu.ppt>
- LeBlanc, M. (2006). Tools for Creating Dramatic Game Dynamics. In K. Salen, & E. Zimmerman (Eds.) *The Game Design Reader: A Rules of Play Anthology*, (pp. 438–459). Cambridge, MA: The MIT Press.
- Librande, S. (2010). One-Page Designs. Presentation at the Game Developers Conference, San Francisco CA, March 2010.
URL <http://stonetronix.com/gdc-2010/>
- Lindley, C. (2003). Game Taxonomies: A High Level Framework for Game Analysis and Design. *Gamasutra*.
URL http://www.gamasutra.com/view/feature/2796/game_taxonomies_a_high_level_.php
- Lister, M., Dovey, J., Giddings, S., Grant, I., & Kelly, K. (2003). *New Media: A Critical Introduction*. Abingdon, UK: Routledge.
- Locke, J. (1975). *An Essay Concerning Human Understanding*. Oxford, UK: Oxford University Press.

- Malaby, T. M. (2007). Beyond Play: A New Approach to Games. *Games and Culture*, (2), 95–113.
- Martin, P. (2011). A spatial analysis of the JBA headquarters in Splinter Cell: Double Agent. In *Proceedings of the Foundation of Digital Conference, Bordeaux 2011*, (pp. 123–130).
- McCloud, S. (2000). *Reinventing Comics: How Imagination and Technology Are Revolutionizing an Art Form*. New York, NY: HarperCollins.
- McGuire, M., & Jenkins, O. C. (2009). *Creating Games: Mechanics, Content, and Technology*. Wellesley, MA: A.K. Peters, Ltd.
- Murray, J. (1997). *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. New York, NY: The Free Press.
- Myers, D. (1999a). Simulation, Gaming and the Simulative. *Simulation & Gaming: An International Journal*, 30(4), 482–489.
- Myers, D. (1999b). Simulations as Play: A Semiotic Analysis. *Simulation & Gaming: An International Journal*, 30(2), 147–162.
- Natkin, S., & Vega, L. (2003). Petri Net Modeling for the Analysis of the Ordering of Actions in Computer Games. In *Proceedings of GAME-ON Conference, London UK, November 2003*, (pp. 82–92).
- Nelson, M. J., & Mateas, M. (2007). Towards Automated Game Design. In *Proceedings of AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, Rome Italy, September 2007*, (pp. 626–637).
- Pagulayan, R. J., Keeker, K., Wixon, D., Romero, R. L., & Fuller, T. (2003). User-centered Design in Games. In J. Jacko, & A. Sears (Eds.) *The Human Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, (pp. 882–906). Mahwah, NJ: Lawrence Erlbaum Associates, Inc.
- Panofsky, E. (1960). *Renaissance and Renascences in Western Art*. Stockholm, Sweden: Almqvist & Wiksell.
- Poole, S. (2000). *Trigger Happy: The Inner Life of Videogames*. London, UK: Fourth Estate.
- Propp, V. (1968). *Morphology of the Folktale*. Austin, TX: University of Texas Press.
- Rekers, J., & Schürr, A. (1995). A Graph Grammar Approach to Graphical Parsing. In *Proceedings of the 11th International IEEE Symposium on Visual Languages, Darmstadt Germany, May 1995*, (pp. 195–202).
- Reyno, E. M., & Carsí Cubel, J. A. (2008). Model-Driven Game Development: 2D Platform Game Prototyping. In *Proceedings of the GAME-ON Conference, Valencia Spain, November 2008*.

- Reyno, E. M., & Carsí Cubel, J. A. (2009a). A Platform-Independent Model for Videogame Gameplay Specification. In *Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference, London UK, September 2009*.
- Reyno, E. M., & Carsí Cubel, J. A. (2009b). Automatic Prototyping In Model-Driven Game Development. *ACM Computers in Entertainment*, 7(2).
- Rogers, S. (2010). *Level Up: The Guide To Great Video Game Design*. Chichester, UK: John Wiley & Sons, Ltd.
- Rollings, A., & Adams, E. (2003). *On Game Design*. Berkeley, CA: New Riders.
- Rosenblum, R. (1975). *Modern Painting And The Northern Romantic Tradition, Friedrich to Rothko*. London, UK: Thames and Hudson Ltd.
- Ryan, M.-L. (2001). *Narrative as Virtual Reality: Immersion and Interactivity in Literature and Electronic Media*. Baltimore, MD: The John Hopkins University Press.
- Saint-Exupéry, A., de (1939). *Wind, Sand and Stars*. London, UK: Heinemann.
- Salen, K., & Zimmerman, E. (2004). *Rules of Play: Game Design Fundamentals*. Cambridge, MA: The MIT Press.
- Saussure, F., de (1983). *Course in General Linguistics*. Chicago, IL: Open Court Publishing Company.
- Schaffer, N. (2008). Heuristic Evaluation of Games. In K. Isbister, & N. Schaffer (Eds.) *Game Usability: Advice from the Experts for Advancing the Player Experience*, (pp. 78–89). Burlington, MA: Morgan Kaufman Publishers.
- Schell, J. (2008). *The Art of Game Design: a book of lenses*. Burlington, MA: Morgan Kaufman.
- Selic, B. (2003). The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5).
- Selic, B., & Rumbaugh, J. (1998). Using UML for Modeling Complex Real-Time Systems. Retrieved September 8, 2008.
URL http://www.ibm.com/developerworks/rational/library/content/03July/1000/1155/1155_umlmodeling.pdf
- Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, 41(314), 256–275.
- Sheffield, B. (2007). Defining Games: Raph Koster's Game Grammar. *Gamasutra*.
URL http://www.gamasutra.com/view/feature/1979/defining_games_raph_kosters_game_.php

- Simpson, Z. (2000). The In-Game Economics of Ultima Online. Presentation delivered at the Game Developers Conference, San Jose, CA, March 2000.
URL <http://www.mine-control.com/zack/uoecon/uoecon.html>
- Smelik, R., Turenel, T., Kraker, K. J., de, & Bidarra, R. (2010). Integrating procedural generation and manual editing of virtual worlds. In *Proceedings of the Foundations of Digital Games Conference Monterey CA, June 2010*.
- Smith, G., Treanor, M., Whitehead, J., & Mateas, M. (2009). Rhythm-Based Level Generation for 2D Platformers. In *Proceedings of the International Conference on the Foundations of Digital Games, Orlando FL, April 2009*, (pp. 175–182).
- Smith, G., Whitehead, J., & Mateas, M. (2010). Tanagra: A Mixed-Initiative Level Design Tool. In *Proceedings of the Foundations of Digital Games Conference, Monterey CA, June 2010*, (pp. 209–216).
- Smith, H. (2001). The Future of Game Design: Moving Beyond Deus Ex and Other Dated Paradigms.
URL http://www.igda.org/articles/hsmith_future.php
- Stephenson, N. (1995). *The Diamond Age, or, A Young Lady's Illustrated Primer*. New York, NY: Bantam Spectra.
- Stevens, P., & Booley, R. (1999). *Using UML: Software engineering with objects and components, updated edition*. Harlow, UK: Addison Wesley Longman Ltd.
- Stiny, G., & Gips, J. (1972). Shape Grammars and the Generative Specification of Painting and Sculpture. In *Proceedings of Information Processing 71*, (pp. 125–135).
- Swain, C. (2008). Master Metrics: The Science Behind the Art of Game Design. In K. Isbister, & N. Schaffer (Eds.) *Game Usability: Advice from the Experts for Advancing the Player Experience*, (pp. 119–140). Burlington, MA: Morgan Kaufman Publishers.
- Sweetser, P. (2006). *An Emergent Approach to Game Design - Development and Play*. Ph.D. thesis, The University of Queensland.
- Sweetser, P., & Wyeth, P. (2005). GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CEI)*, 3(3).
- Taylor, M. J., Gresty, D., & Baskett, M. (2006). Computer Game-Flow Design. *ACM Computers in Entertainment*, 4(1).
- Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). What is Procedural Content Generation? Mario on the borderline. In *Proceedings of the Foundations of Digital Games Conference, Bordeaux France, June 2011*.
- Togelius, J., Preuss, M., & Yannakakis, G. N. (2010). Towards multiobjective procedural map generation. In *Proceedings of the Foundations of Digital Games Conference, Monterey CA, June 2010*.

- Veugen, C. (2011). *Computer Games as Narrative Medium*. Ph.D. thesis, Vrije Universiteit Amsterdam.
- Vogler, C. (2007). *The Writer's Journey: Mythic Structure for Writers*. Studio City, CA: Michael Wiese Productions, third ed.
- Ward, J., Cantor, D., & Carey, B. (2007). The Hard Science of Making Videogames.
URL <http://www.popsoci.com/popsoci/technology/d997f0209dd15110vgnvcm1000004eecbccdrd.html>
- Wardrip-Fruin, N. (2009). *Expressive Processing*. Cambridge, MA: The MIT Press.
- Wardrip-Fruin, N., Mateas, M., Dow, S., & Sali, S. (2009). Agency Reconsidered. In *Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference, London UK, September 2009*, (pp. 1–9).
- Wolf, M. J. P. (2001). The Video Game as a Medium. In M. J. P. Wolf (Ed.) *The Medium of the Video Game*, (pp. 13–33). Austin, TX: University of Texas Press.
- Wolfram, S. (2002). *A New Kind of Science*. Champaign, IL: Wolfram Media Inc.
- Woods, W. A. (1970). Transition Network Grammars for Natural Language Analysis. *Computational Linguistics*, 13(10), 591–606.
- Wright, W. (2003). Dynamics for Designers. Presentation at the Game Developers Conference, San Jose CA, March 2003.
URL <http://www.slideshare.net/geoffhom/gdc2003-will-wright-presentation>
- Zagal, J. P., Mateas, M., Fernández-Vara, C., Hochhalter, B., & Lichti, N. (2005). Towards an Ontological Language for Game Analysis. In *Changing Views - Worlds in Play: Proceedings of the 2009 Digital Games Research Association Conference, Vancouver Canada, June 2005*.

Ludography

- Agricola (2007, board game). Lookout Games, Uwe Rosenberg.
- Americas Army (2002, PC). U.S. Army.
- Angry Birds (2009, iOS, Android, others). Rovio Mobile Ltd.
- Ascent (2009, PC). J. Dormans.
- Assassins Creed (2007, PS3, Xbox 360). Ubisoft, Inc., Ubisoft Divertissements Inc.
- Baldurs Gate (1998, PC). Interplay Productions, Inc. BioWare Corporations.
- Bejeweled (2000, PC). PopCap Games, Inc.
- Bewbees (2011, Android, iOS). GewGawGames.
- Blood Bowl, third edition (1994, board game). Games Workshop Ltd, J. Johnson.
- Boulder Dash (1984, Apple II, Commodore 64, NES, others). First Star Software, Inc.
- Caesar III (1998, PC). Sierra On-Line, Inc., Impressions Games.
- Campagne (2010, card game). M. van Dodeweerd & J. Dormans.
- Caylus (2005, board game). White Goblin Games, William Attia.
- Civilization (1991, PC). MicroProse Software, MPS Labs.
- Civilization III (2001, PC). Infogrames Europe SA, Firaxis Games.
- Colossal Cave Adventure (1975-76, PDP-10) W. Crowther & D. Woods.
- Command & Conquer: Red Alert (1996, PC, PS). Virgin Interactive Entertainment (Europe) Ltd., Westwood Studios, Inc.
- Connect Four (1974, board game). Milton Bradley, N. Strongin & H. Wexler.
- Counter-Strike (2000, PC). Sierra On-Line, Inc., Valve Corporation.
- Descent: Journeys in the Dark (2005, board game). Fantasy Flight, K. Wilson.
- Deus Ex (2000, PC, PS2). Eidos Interactive Ltd., Ion Storm Inc.
- Diablo (1996, PC, PS). Blizzard Entertainment Inc., Blizzard North.
- Donkey Kong (1981, arcade). Nintendo Co., Ltd.

- Dungeon Run (2009, PC). Joris Dormans.
- Dungeons & Dragons (1974, pen-and-paper RPG). Tactical Study Rules, Inc., G. Gygax & D. Arneson.
- Elite (1984, Apple II, Commodore 64, PC, others). Acornsoft Limited.
- EverQuest (1999, PC). 989 Studios, Verant Interactive.
- Eye of the Beholder II (1991, PC, Amiga). Strategic Simulations, Inc., Westwood Associates.
- Fable (2004, XBox). Microsoft Game Studios, Lionhead Studios Ltd.
- Fallout 3 (2008, PC, PS3, Xbox360). Bethesda Softworks LLC, Bethesda Game Studios.
- Fearsome Floors (2003, board game). 2F-Spiele, F. Friese.
- Flix (2009, PC). J. Dormans.
- Flix 2 (2010, PC). J. Dormans.
- Get H2O (2010, board game). 999 Games, Butterfly Works, Mama Bits & J. Dormans.
- Grand Theft Auto III (2001, PS2, PC). Rockstar Games, Inc., DMA Design Limited.
- Grand Theft Auto: San Andreas (2004, PS2, PC, Xbox). Rockstar Games, Inc., Rockstar North Ltd.
- Half-Life (1998, PC). Sierra On-Line, Inc., Valve L.L.C.
- Half-Life 2 (2004, PC, XBox). Sierra Entertainment, Inc., Valve Corporation.
- Halo (2001, PC, XBox). Microsoft Corporation, Bungie Studios.
- Infinite Boulder Dash (2011, PC). J. Dormans.
- Infinite Mario Bros (2006, PC). M. Persson.
- Knytt Stories (2007, PC). N. Nygren.
- Kriegsspiel (1824, board game). B. von Reisswitz.
- Little Big Planet (2008, PS3). Sony Computer Entertainment America, Inc., Media Molecule Ltd.
- Magic the Gathering (1993, trading card game). Wizards of the Coast, R. Garfield.

- MineCraft (2010, PC). Mojang AB.
- Monopoly (1934, board game). Parker Brothers.
- Pac-Man (1981, arcade). Namco Limited.
- Portal (2007, PC). Buka Entertainment, Valve Corporation.
- Power Grid (2004, board game). 2F-Spiele, F. Friese.
- Prince of Persia (1989, DOS, others). Brøderbund Software, Inc.
- Prince of Persia: Sands of Time (2003, PC, PS2, Xbox, GameCube). Ubisoft, Inc., Ubisoft Divertissements Inc.
- Puerto Rico (2002, board game). Alea, A. Seyfarth.
- Risk (1959, board game). Parker Brothers, A. Lamorisse & M.I. Levin.
- Robot Wants Kitty (2010, PC). MaxGames.com, Hamumu Software.
- Rogue (1983, PC, others). Artificial Intelligence Design.
- Seasons (2010, PC). J. Dormans.
- September 12 (2003, PC). Newsgaming.com, G. Frasca.
- Settlers of Catan (1995, board game). Franckh-Kosmos Verlag, K. Teuber.
- SimCity (1989, Amiga, Commodore 64, PC, others). Infogrames Europe SA, Maxis Software Inc.
- Space Hulk (1989, board game). Games Workshop Ltd.
- Space Invaders (1980, arcade). Taito Corporation.
- Spore (2008, PC). Electronic Arts, Inc., Maxis Software Inc.
- SPUG (2009, PC). Electronic Arts, Inc., Maxis Software Inc.
- Star Defender (2002, PC). Awem Studio.
- Star Wars: X-Wing Alliance (1999, PC). LucasArts Entertainment Company L.L.C., Totally Games, Inc.
- StarCraft (1998, PC). Blizzard Entertainment Inc.
- StarCraft II (2010, PC). Blizzard Entertainment Inc.
- Super Mario Bros. (1985, NES). Nintendo Co. Ltd.
- Super Mario Kart (1992, SNES). Nintendo Co. Ltd.

Super Mario Sunshine (2002, GameCube). Nintendo Co. Ltd., Nintendo EAD.

System Shock 2 (1999, PC). Electronic Arts, Inc., Looking Glass Studios, Inc.

Tetris (1986, PC, others). AcademySoft, A. Pazhitnov.

The Elder Scrolls IV: Oblivion (2006, PS3, PC, Xbox360). Bethesda Softworks LLC, Bethesda Game Studios.

The Landlord Game (1904, board game). E. Magie.

The Legend of Zelda (1986, NES). Nintendo Co. Ltd.

The Legend of Zelda: Twilight Princess (2006, GameCube, Wii). Nintendo Co. Ltd., Nintendo EAD.

The Sims (2000, PC, PS2, Xbox, GameCube). Electronic Arts, Inc., Maxis Software Inc.

Torchlight (2009, PC, Xbox360). Runic Games, Inc.

Up the River (1988, board game). Ravensburger, M. Ludwig.

Warhammer Fantasy Roleplay (1986, pen-and-paper RPG). Games Workshop.

World of Goo (2008, PC, Wii, iOS). 2D Boy.

Zork (1979, PDP-10). T. Anderson, M. Blank, B. Daniels, & D. Lebling.