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 divers. 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 build, 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.

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

[2]The developers of reusable game components that are sold to game studies ('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.

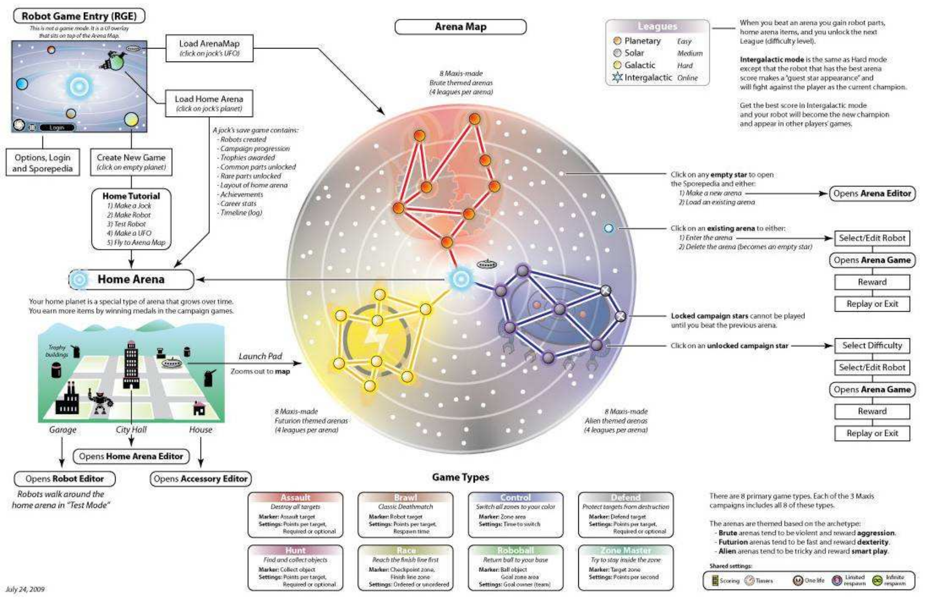
Robot Game Entry (RGE)
This is not a game mode. It is a UI overlay that sits on top of the Arena Map.
Load ArenaMap (click on jock's UFO)
Load Home Arena (click on jock's planet)
Options, Login and Sporepedia
Create New Game (click on empty planet)
Home Tutorial
1) Make a Jock
2) Make Robot
3) Test Robot
4) Make a UFO
5) Fly to Arena Map
A jock's save game contains:
- Robots created
- Campaign progression
- Trophies awarded
- Common parts unlocked
- Rare parts unlocked
- Layout of home arena
- Achievements
- Career stats
- Timeline (log)
Home Arena
Your home planet is a special type of arena that grows over time.
You earn more items by winning medals in the campaign games.
Trophy buildings
Launch Pad
Zooms out to map
Garage
City Hall
House
Opens Home Arena Editor
Opens Robot Editor
Robots walk around the home arena in "Test Mode"
Opens Accessory Editor
Arena Map
8 Maxis-made Brute themed arenas (4 leagues per arena)
8 Maxis-made Futurion themed arenas (4 leagues per arena)
8 Maxis-made Alien themed arenas (4 leagues per arena)
Leagues
Planetary Easy
Solar Medium
Galactic Hard
Intergalactic Online
When you beat an arena you gain robot parts, home arena items, and you unlock the next League (difficulty level).
Intergalactic mode is the same as Hard mode except that the robot that has the best arena score makes a "guest star appearance" and will fight against the player as the current champion.
Get the best score in Intergalactic mode and your robot will become the new champion and appear in other players' games.
Click on any empty star to open the Sporepedia and either:
1) Make a new arena
2) Load an existing arena
Opens Arena Editor
Click on an existing arena to either:
1) Enter the arena
2) Delete the arena (becomes an empty star)
Select/Edit Robot
Opens Arena Game
Reward
Replay or Exit
Locked campaign stars cannot be played until you beat the previous arena.
Click on an unlocked campaign star
Select Difficulty
Select/Edit Robot
Opens Arena Game
Reward
Replay or Exit
Game Types
Assault
Destroy all targets
Brawl
Classic Deathmatch
Control
Switch all zones to your color
Defend
Protect targets from destruction
Hunt
Find and collect objects
Race
Reach the finish line first
Roboball
Return ball to your base
Zone Master
Try to stay inside the zone
There are 8 primary game types. Each of the 3 Maxis campaigns includes all 8 of these types.
The arenas are themed based on the archetype:
- Brute arenas tend to be violent and reward aggression.
- Futurion arenas tend to be fast and reward dexterity.
- Alien arenas tend to be tricky and reward smart play.
Shared settings:
July 24, 2009

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

4. 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

[3]Unfortunately 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.

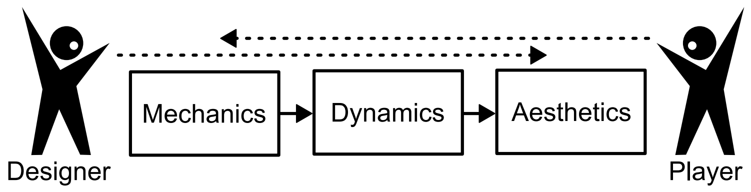

Figure 3.2: The MDA framework (after Hunicke et al., 2004).

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 where 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

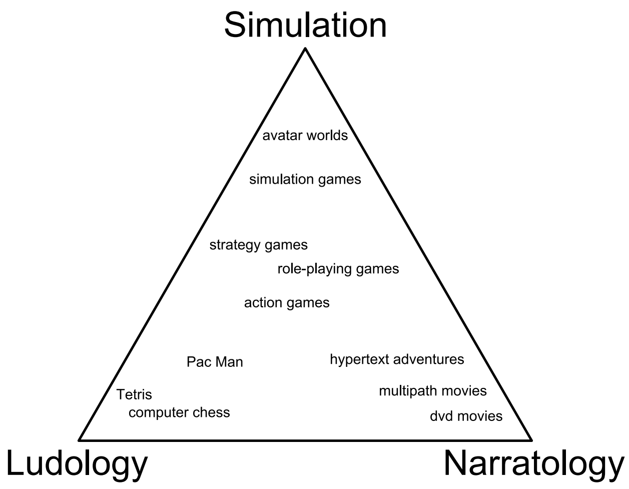

Figure 3.3: One of Craig Lindley's taxonomies (after Lindley, 2003).

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

[4]`http://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.

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

[6]The 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 transfered 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

[7]Also know 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 reoccur 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

[8]Although 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.

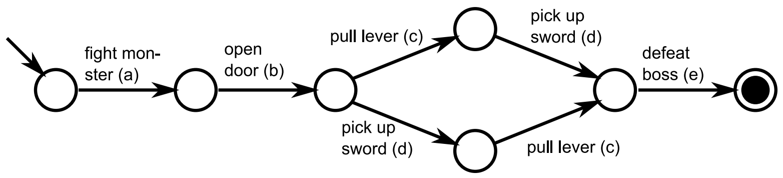

Figure 3.4: A finite state machine representing an adventure game.

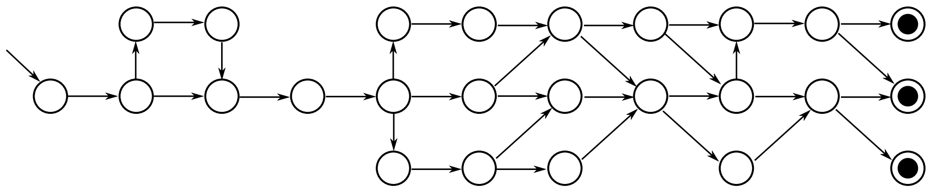

Figure 3.5: A more complex finite state machine, but one that still produces a finite set of trajectories.

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,*

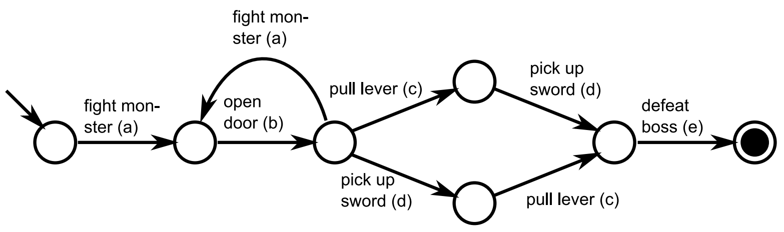

Figure 3.6: A looping finite state machine representing an adventure game.

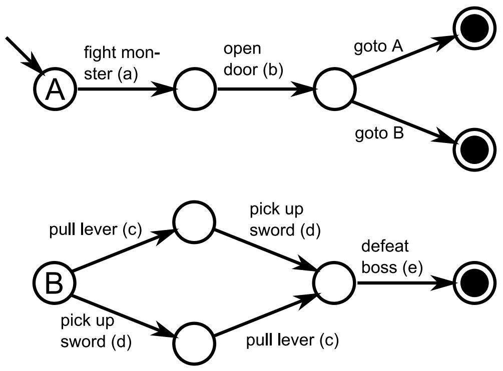

Figure 3.7: A recursive finite state machine representing an adventure game.

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-

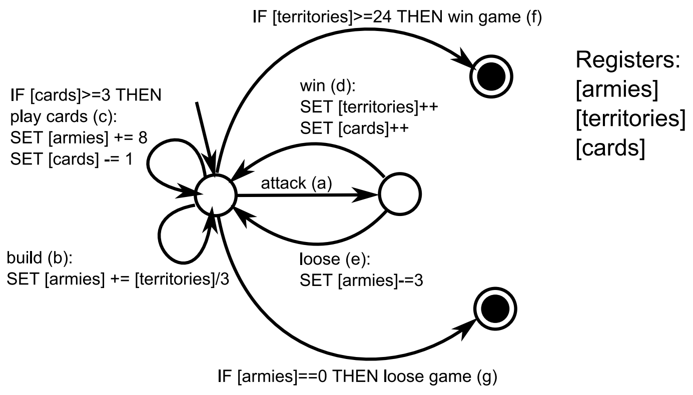

Figure 3.8: A state diagram for Risk

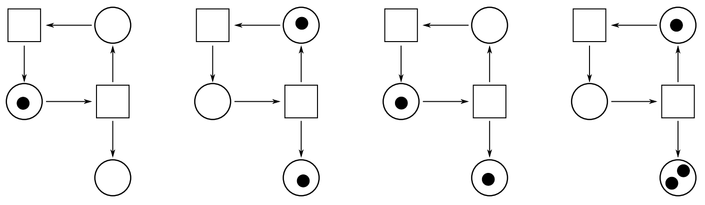
Figure 3.9: Four iterations of the same Petri net showing movement of tokens through the network

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-

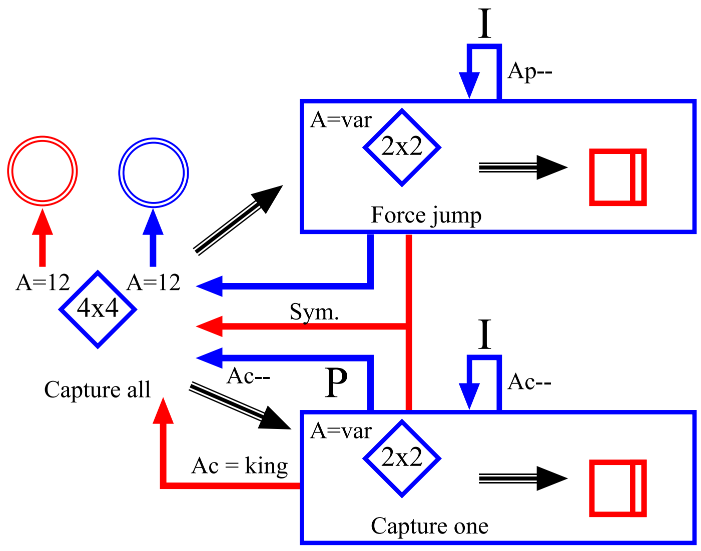

Figure 3.10: Raph Koster's diagram of CHECKERS (2005a).

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

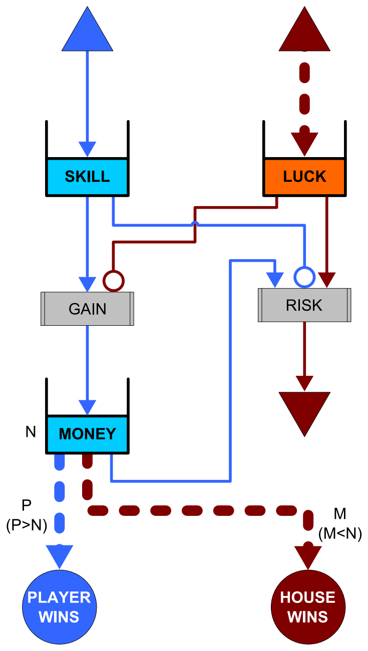

Figure 3.11: Stéphane Bura's diagram of Blackjack. (2006)

(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

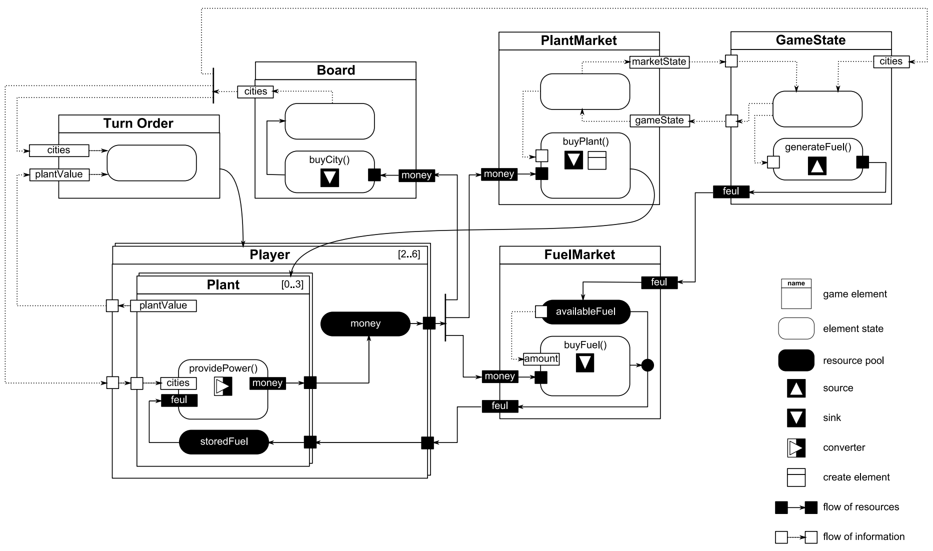

Figure 3.12: Power Grid in an adaptation of UML collaboration diagrams.

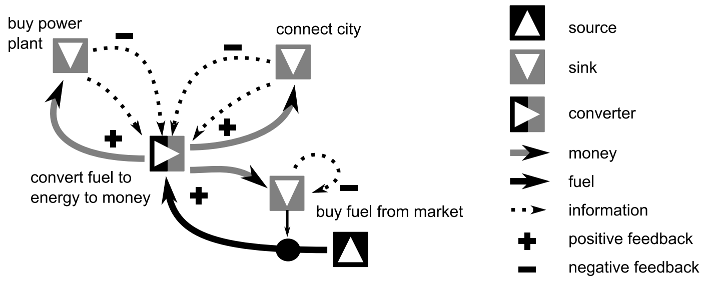

Figure 3.13: The feedback structure in Power Grid.

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

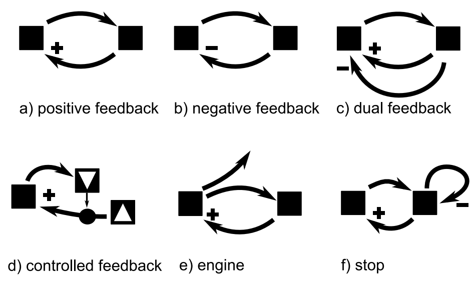

Figure 3.14: The feedback patterns extracted from POWER GRID.

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.

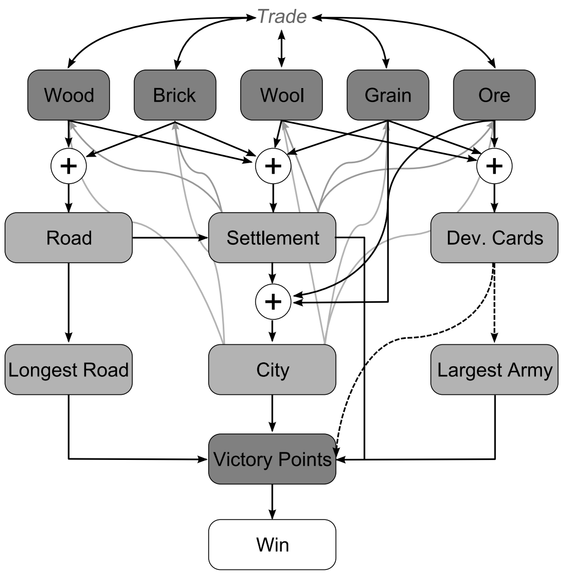

Figure 3.15: Commodity flow graph for SETTLERS OF CATAN, after McGuire & Jenkins (2009, 243).

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 designers 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.