Jussi Markus Kuittinen

Computer-Aided Game Design

Master's Thesis
in Information Technology
January 19, 2008

University of Jyväskylä

Department of Mathematical Information Technology

Jyväskylä

Author: Jussi Markus Kuittinen

Contact information: jussi.kuittinen@uta.fi

Title: Computer-Aided Game Design

Työn nimi: Tietokoneavusteinen pelisuunnittelu

Project: Master's Thesis in Information Technology

Page count: 81

Abstract: Games are getting to be more and more important part of the entertainment industry of the world, yet methodological game design approaches are still not very widely researched. A computer-aided game design and analysis tool could provide better comprehension, communication and documentation of game design projects, but such a tool requires a structured and rule-based way of describing a game design. This master's thesis presents a conceptual model understood as a structure of games' meaning-producing elements, where the elements and their inter-relations are described using game design patterns. The model serves as a basis for a software game design and analysis tool called CAGE, which allows designers and researchers to visualise game designs as conceptual models in manner similar to CASE-tools in software engineering. The tool shows initial promise and amounts to a better understanding of the subject, but leaves some questions concerning the suitability of the chosen approach unanswered and in need of further research.

Suomenkielinen tiivistelmä: Pelit ovat jatkuvasti tärkeämpi osa koko maailman viihdeteollisuutta, mutta niiden metodologiset suunnittelumenetelmät ovat yhä suhteellisen vähän tutkittuja. Tietokoneavusteinen pelisuunnittelu- ja analysointityökalu mahdollistaa paremman tavan kommunikoida ja dokumentoida pelisuunnitelmia, mutta vaatii rakenteisen ja sääntöpohjaisen tavan pelisuunnitelmien kuvaamiseen. Tässä Pro Gradu-tutkielmassa esitellään käsitemalli peleistä merkityksiä tuottavien elementtien rakenteena, jossa elementit ja niiden väliset suhteet kuvataan pelisuunnittelumalleilla (game design patterns). Tämä käsitemalli on pohjana pelisuunnittelu ja analysointiohjelmistolle nimeltä CAGE, jonka avulla suunnittelijat ja tutkijat voivat visualisoida pelisuunnitelmia samaan tapaan kuin ohjelmistotuotannossa käytettyjen CASE-työkalujen avulla. CAGE herättää lupauksia ja lisää ymmärrystämme aihealueesta, mutta osaan kysymyksiä koskien valitun lähestymistavan sopivuutta, ei voida vastata täydellisesti,

vaan aihe vaatii lisätutkimusta.

Keywords: Game design, game design patterns, computer-aided design, CASE-tools

Avainsanat: Pelisuunnittelu, pelisuunnittelumallit, tietokoneavusteinen suunnittelu, CASE-työkalut

Preface

This work has been conducted as part of an european games research project, the Intergrated Project on Pervasive Gaming (FP6-004457), which is funded under the European Comission's IST Programme. The Master's Thesis itself was comissioned by Nokia Research Center.

After a very long process, I can only wonder at the seemingly endless patience of my wife Anna, and my supervisors, Jonne Itkonen from the University of Jyväskylä and Jussi Holopainen from Nokia Research Center. Thank you indeed.

Contents

Preface **i**

1 Introduction **1**

2 Elements of game design **4**

2.1 The structure of game design 4

2.2 Rules 5

2.2.1 Qualities of rules 5

2.2.2 Game as a system 9

2.2.3 Rules in games 11

2.2.4 Emergence and progression 16

2.3 Constituents of meaning 18

2.3.1 Meaning and interaction 18

2.3.2 Goals and actions 20

2.3.3 Opposition and conflict 22

2.3.4 Information 22

2.3.5 Representation and narratives 25

2.3.6 Social interaction 26

3 Methodological Game Design **29**

3.1 Introduction 29

3.2 Defining the Game Design Method 29

3.3 General description of game design process . . . 33
3.4 Game design tools and methods . . . 34
3.4.1 A model to support the design of multiplayer games (Nussbaum et al.) . . . 34
3.4.2 Mechanics-Dynamics-Aesthetics . . . 36
3.4.3 The 400 Project . . . 38
3.4.4 Game Design Workshop by Fullerton, Swain and Hoffman . . . 39
3.4.5 GameGame . . . 40
3.4.6 Player-Centred Design . . . 41
3.5 Game design patterns . . . 42
3.5.1 Related approaches . . . 42
3.5.2 Characteristics of game design patterns . . . 43
3.5.3 Uses of game design patterns . . . 45

4 Computer-aided game design **47**
4.1 Computer-aided software engineering . . . 47
4.2 Conceptual modelling of game design . . . 49
4.3 Structure of *CAGE* . . . 53
4.3.1 Higher level structure . . . 53
4.3.2 *sdf*-package . . . 55
4.3.3 *opengl*-package . . . 60
4.3.4 *cage*-package . . . 62
4.4 Case study: *Unintelligent Designer* . . . 64

5 Conclusions and further study **69**

6 Bibliography **71**

1 Introduction

During the last ten years at least, digital games have become a major part of the entertainment industry of the world. In the United States alone, the revenues from video and computer games were 8.2 billion dollars in year 2004 and according to a report by the Entertainment Software Association, are expected to raise to 15 billion dollars in year 2006 (Crandall and Sidak, 2006, 2-3). Digital games industry also uses significant proportions of their revenue on research and development: the same report estimates it to be around 19 percent (Crandall and Sidak, 2006, 7).

Although the investment in R&D seems great, it does not necessarily translate to great investment in basic research on game development, but instead appears to concentrate more on developing advanced equipment and implementation technologies. It is true that video games are very technology-driven and that games compete with their technological advancement, but this has maybe sidetracked games research somewhat. Serious, scientific literature on the design and structure of games is quite scarce as can also be seen in this thesis.

As pointed by Björk et al. (2007), turning the tacit knowledge on game design into explicit knowledge is not a trivial issue for a number of reasons, of which the required effort is not the least. Game design is essentially experience design, where the way players create meaning within the game context, i.e. experience the play, defines the successfulness of the game design. Therefore the task of the designer is to anticipate players' responses to different situations within the game and try to use them to make the game a good experience for the players.

At the heart of the game design is the notion of *rule*. Discussing the concept of *game*, Salen and Zimmerman (2004, 79) list eight definitions by influential game scholars David Parlett, Clark Abt, Johann Huizinga, Roger Callois, Bernard Suits, Chris Crawford, Greg Costikyan and finally, by Elliot Avedon and Brian Sutton-Smith. The simplified chart by Salen and Zimmerman, showing all the elements of games as defined by the authors and how common these elements are in the definitions, verifies the cen-

trality of rules in games. The definitions include elements such as *goals*, *conflict*, *decision making*, *not serious*, *voluntary*, *resources* and *artificial*, but only one element, rules, is common to all but one author[1].

As will be discussed in depth in chapter 2.2 games are really systems of rules so the designer is actually anticipating how players respond to a set of rules. One key argument in this thesis is that these rules in turn and the way players respond to them, are defined and affected by numerous rules and rule-like guidelines which are not part of the game rules. Therefore, the designer's task is to create a system of rules, where the effects of those external rules are minimised and controlled.

The research problem leading to this thesis was initially to study how one could make a software tool for designing games and how game design patterns could be utilised in such a program. It soon became clear, that in order to make most use of a design tool, one needs a methodological approach to design, where the design process can be iterated for refinement and documented for learning from one's choices. Common design tools in software engineering, CASE-tools, were chosen as models for the tool to be made. As a consequence, a way of creating an abstracted structural drawing of a game design was required.

The view purported in this work treats game design as experience design, where it is necessary to understand how this experience is built and what are the factors that affect it. The theoretical foundation for this thesis is basically two-fold:

1. A design of a game can be seen as an inter-related structure of *meaning-producing elements*, such as conflict between the players, representation, goals and actions, etc. Although there is a selected set of elements introduced in this work, it should be stressed that this set is by no means meant as the one and only set, but only as one possibility.

2. There are *rule-like principles* for estimating and explaining the effects of using and combining these meaning-producing elements. Game design patterns (see 3.5) act as such a set of principles for gameplay.

[1] Although Costikyan didn't include rules as a key element in games in his essay 'I have No Mouth and I Must Design' (Costikyan, 1994), it is quite evident that he does not see rules as something not belonging to games.

The resulting game design and analysis software, *CAGE*, builds upon this foundation. The tool aids designers and researchers in analysing existing game designs by defining a set of structural elements, which can be used to describe a game design as a collection of design choices and especially the causes and consequences of these choices. The product of using *CAGE* for analysis or design is a visual graph, which depicts the game design as inter-connected, documented and discernable meaning-producing elements. Each element and the way they connect to other elements are described by using game design patterns.

Chapter 2 describes the theoretical background for understanding the relation between game rules and the rule-like principles for creating and affecting player experiences. The selected set of meaning-producing elements and their justification is also given in here. Chapter 3 first discusses the idea of methodological game design and introduces some of the current method-like game design theories. Finally an in-depth view on game design patterns is given and their importance for the chosen game design method is established. In Chapter 4, the design decisions and requirements for *CAGE* are explained and an example case for using the program is given. Lastly, Chapter 5 offers the concluding remarks and some considerations for future research on the subject.

2 Elements of game design

2.1 The structure of game design

The key to understanding game design is to understand its purpose as *experience design*. Everything a game designer does is aimed for creating meaningful experiences for the players of the game. Compared to filmmakers or writers, for instance, the tool arsenal in game design is quite much larger. Like them, game designers too can use narrative or cinematic techniques to create different kinds of experiences, but in addition they can also utilise more efficiently such things as social interaction between the players and the dynamic nature of games allowing players to actually influence the way the game is played. Naturally all this makes game design also a very complex process.

The one building block a game designer has is the concept of *rule*. The view that is central to understanding this master's thesis is that games are simply systems of rules and that game design is essentially rule design. Of course a single game will consist of and be affected by so many rules, that it would be fruitless to start thinking game design as creating individual rules one after the other, but it is still important to understand that the structure producing meaningful experiences is still just a structure of rules.

This chapter will focus on examining two theoretical notions forming the basis upon which chapters three and four will build:

1. Firstly the idea of structure for game designs. Similar to dramatic structure in literary theory, games also have structure, but its composition and elements are different. In games there are various kinds of *meaning producing elements* such as goal structures, conflicts, social interaction and so on forth. This thesis builds upon the idea that game designs can be structured as an inter-related set of meaning producing elements.

2. Secondly, the *principles of game design* can be used to describe the relations between meaning producing elements. The concept is borrowed from jurispru-

dence, where *legal principles* (Hart, 1997, 260-261) define justification and background to legal rules so that in cases where a law does not explicitly state a result, it can be reached by studying the principles justifying the rule. In a similar way, the principles of game design provide justification to game rules by pointing out general uses and effects of different rule-structures. As these principles are rather general descriptions of ideals and practices, they are easier to use than individual rules. A simple example of a principle of game design would be for instance "Rewards linked to risks heighten the player's tension and create more meaningful decision making situations."

Although the principles of game design will be examined in more detail as game design patterns in section 3.5, the purpose of this chapter is to provide the justification and the basis for understanding them. Hence the lenghty and tedious discussion on rules and their role in games. The notion emphasised here is that games should not be viewed as formal systems of unambiguous and clear rules, but that it is at more important to understand the reasons how a given set of rules will be understood by the players and this in turn is dependent at least on the social, cultural and psychological contexts surrounding the game system. The principles behind rules are what explain these contexts and provide rule-like guidelines for using them. The constituents of meaning described at the end of this chapter in section 2.3 offer a one possible, gameplay-centric way of classifying these principles.

2.2 Rules

2.2.1 Qualities of rules

A concept so commonly applied to games, 'rule' is certainly among the more perplexing notions in games research. At first sight it seems easy enough: a rule defines some procedure happening over the course of gameplay, e.g. '*Player with white pieces begins the game*' or '*A bishop moves diagonally*'. For a player, rules describe how to play the game and are something that comes with the game, maybe in a booklet found in the game's packaging or maybe explained by other players before or during the actual instance of

gameplay. Without knowing the rules, it would be impossible to play the game. Rules are what make games into games. What exactly then are rules?

Collins English Dictionary (1991, 1353) provides the following as one possible definition:

> "1. an authoritative regulation or direction concerning method or procedure, as for court of law, legislative body, game or other human institution or activity: *judges' rules; play according to the rules.*"

This offers good insight into rules: they are authoritative directives and concern human procedures. It is also the way rules are commonly understood: a *rule prescribes or proscribes human behaviour and is seen as a binding order*. In the case of the two rule examples previously stated, both are prescriptive rules giving directions on how to do something. In this particular case, one could also state them as proscriptive rules, e.g. '*Player with non-white pieces cannot begin the game*', but there wouldn't be much point.

Although not explicit in the definition, one can also deduce from it that rules are *contextual*: they are applied only in predetermined situations, which in this case are the court of law, legislative body and game. These are still very broad contexts, normally rules are more specifically defined. For example, looking again at the two rule examples in the first paragraph, one can see that they are both rules of *Chess*, so they share a common context: instance of a game of *Chess*, but this is still quite broad. Their actual contexts are more clear cut than that: the first rule is only applied when beginning the game and the second rule only when a player moves the bishop.

Rules also have a *standing nature* (Hart, 1997, 23): a valid rule is applied over and over again if its field of application is reached. This accounts to an important aspect of rules: degree of predictability (Hart, 1997, 147). If a rule is known to players and it is clearly defined, then all players can expect it to be applied every time it is applicable. In games, predictability makes two essential things possible: it provides a basis for strategy and allows games to be replayable. The degree of predictability varies according to the clarity of the rule - a rule with many possible interpretations is harder to predict - but can never be totally absent. A rule with no predictable consequences can simply not be a rule.

The definition from the Collins English Dictionary raises one important question: where does a rule get its authorization? This is a central problem in jurisprudence, but for the purposes of this subject, it suffices to say that a rule can get its authority from other rules and ultimately, from a process of acceptance[1] within a society, i.e. the rule's subjects accept the rule as binding and adjust their behaviour accordingly (Dworkin, 1996, 20).

So, a rule is contextual, has a standing nature, controls behaviour and is seen as binding if its authority is accepted. What about rule-breaking then? Clearly there are always some people that do not play by the rules all the time. This leads to another aspect of rules: their authority has to be enforced. If breaking a rule does not lead to any consequences, a rule can hardly be seen as binding. At the slightest these consequences can be anything from frowning upon the rule-breakers or maybe even some name calling. At worst rule breaking can lead to serious results, although in games this does not normally mean the death of the actual player, more likely just the player's character in the game.

Rule-breaking can also be unintended: one can imagine numerous situations where the intent of the rule is misunderstood or the field of its application can not be determined accurately enough. Rules are supposed to be explicit and unambiguous (Salen and Zimmerman, 2004, 123), but that is hardly always the case. Rules comprise of concepts, which are not necessarily clearly defined even within the context they are used. Consider for example a rule in the official laws of *Chess*, FIDE Laws of Chess (2004):

> 12.6 It is forbidden to distract or annoy the opponent in any manner whatsoever. This includes unreasonable claims or unreasonable offers of a draw.

What constitutes as distracting? Breathing heavily? Repeatedly poking the other player with a sharp stick? The latter for sure, but the former? If done purposely to distract, then it would probably be against the rule, but how is one to distinguish purposeful distraction from a, say, natural response to excitement, which it also could be? It is clearly impossible to state this rule's context of application explicitly and unambiguously, because first of all, one simply cannot list all possible forms of distracting behaviour since

[1] See for example H.L.A. Hart's 'rule of recognition'.

there could be thousands. Secondly, being distracted or annoyed at something is a very individual emotion and its causes are very hard to root out. This is why FIDE rulebook states in its preface that

> The Laws of Chess cannot cover all possible situations that may arise during a game, nor can they regulate all administrative questions. Where cases are not precisely regulated by an Article of the Laws, it should be possible to reach a correct decision by studying analogous situations, which are discussed in the Laws. The Laws assume that arbiters have the necessary competence, sound judgment and absolute objectivity. (FID, 2004)

Interestingly enough, the previous excerpt isn't a valid rule in *Chess*, it is just hoped that all players and federations accept this view. So, how then are these rules of ambiguous nature enforced? In one school of thought in legal theory such rules are called *legal principles*, which according to Hart differ from *legal rules* by being "general or unspecific", "explanative or justificatory in relation to rules", and to a degree, non-conclusive in that they do not "necessitate a decision, but point towards or count in favour of a decision" (Hart, 1997, 260-261). As an example, if the previous excerpt were made a rule, then in the case of rule 12.6 in Laws of Chess, the arbiter would be seen as the authority in interpreting the 'analogous situations', which in turn would constitute as the legal principles of *Chess*. This still does not make the rule 12.6 explicit and unambiguous, but it provides a basis of justification for the rule in question, thus making its acceptance and enforcement easier to do.

As a summation, in this thesis rules are said to have the following qualities:

1. Rules are contextual.
2. Applicability of rules can be predicted to a degree.
3. Rules control behaviour.
4. Rules are binding if their authority is accepted.
5. Rules can, but do not have to be explicit and unambiguous.

2.2.2 Game as a system

Johan Huizinga provides an interesting theory on play in his seminal work Homo Ludens. He saw play as an essentially free activity, which was confined from 'normal' life in creating a concretely or abstractly existing, closed space where the rules of play were valid (Huizinga, 1980, 30). This idea of separation is a bit vague in Huizinga's book, but it is probably safe to say that although play creates its own world, the players themselves are not totally free from the surrounding environment. Now, if Huizinga is correct, two important questions follow: 1) how do players know when they have entered the play's space, and 2) in what ways is this space tied to the surrounding world? For Salen and Zimmerman the answer to the first question is: 'players *choose* to play', and to the latter: 'it depends on how the game is framed'.

In their article 'This is not a game - Play in cultural environments' (2003), Salen and Zimmerman study three games that are based on the idea that they try to erase, or at least blur, the boundary between the real world and the game world. As a starting point in their analysis, Salen and Zimmerman provide the concept of *magic circle*, defined in their earlier work (Salen and Zimmerman, 2004, 94). The name magic circle is borrowed from Huizinga and is essentially the same as the confined space outlined by him, but more thoroughly defined. Their eloquent description of the magic circle deserves a quote:

> "As a closed circle, the space it circumscribes is enclosed and separate from the world. As a marker of time, the magic circle is like a clock: it simultaneously represents a path with a beginning and end, but one without beginning and end. The magic circle inscribes a space that is repeatable, a space both limited and limitless. In short, a finite space with infinite possibilities". (Salen and Zimmerman, 2004, 95)

As players enter the magic circle, they begin playing, and once they leave it, they stop playing. As such it seems like all this is just a fancier way of saying 'now I'm playing', but there is more to it. Firstly, the magic circle marks the boundary between the real world and the game world, but it also provides a frame of understanding for the players to know that they are playing (Salen and Zimmerman, 2004, 94). In other words, the key

issue with the magic circle is that players enter into and step out of it willingly, i.e. they *choose* to play, and once they are in it, they must know they are within its boundaries, otherwise they aren't playing a game. Secondly, games have a special property which makes the magic circle more explicit: games have formalized rules known to players (Salen and Zimmerman, 2004, 95). There are certain qualities to these rules, such as their standing nature or repeatability, that actually make up the structure of the magic circle as described in above quote.

In short, the magic circle is a psychological state which, although individual to each player, is supported and re-enforced by the game's rules, which on the other hand, all players share. Simplistically put, players experience a game individually, but all players know that their experience is most likely similar to other players' experience.

Returning to the problem of how players know they are playing a game, Salen and Zimmerman argue that in all three games they studied, players always knew they were playing a game (Salen and Zimmerman, 2003). The boundaries of the magic circle might stretch and bend, but in each case it never completely vanished. Had it dissapeared, one could not have called them games, Salen and Zimmerman conclude. The magic circle is therefore not only a description of a game, but also a prerequirement to a game. Players must enter it voluntarily and it needs to uphold discernible borders so that players can always tell whether they are playing a game or not.

As to the other question of how games are connected to the surrounding environment, Salen and Zimmerman begin their answer by noting that games are essentially systems, where "a set of parts [...] together form a complex whole" (Salen and Zimmerman, 2004, 50). Depending on how these systems are viewed, or framed as Salen and Zimmerman call it, defines the relationships between the game and the real world. If viewed as a *formal system* of rules, the game is a closed system having no contact with the surrounding environment. If seen as an *experiental* system, where the experiences of the players are the key framing factor, games can be either open or closed systems. Finally, if viewed as a *cultural* system, games are always open systems being dependent on the surrounding environment.

All these framings exist simultaneously in all games, Salen and Zimmerman (2004, 52) emphasize, but it is still quite useful to differentiate between them. No game can be played as a closed system - players as humans cannot help but bring in their social and

cultural contexts into the game - yet the game's system of rules can be decontextualised and analysed as a formal system. In the same manner, player experiences can be decontextualised and viewed only in regards to the decisions and choices supported by the closed rule system, but play can also be seen as resulting from players' social and emotional contexts. Finally, a game can be viewed as a cultural entity where the rules, player experiences but also the game itself have cultural meanings.

Although it is often important to view games or at least part of games as closed systems of rules, it may not be always possible. Games such as *Chess* or most card-games, which can modelled mathematically in terms of their core movement and placement rules for instance, can be studied as closed systems. However, if one accepts that rules are not always unambiguous or have a clear field of application, the closed system framing becomes a specialty case. Therefore, it is all the more important for the designer to understand games as systems affected by the player's social, cultural, emotional and psychological contexts. Knowing how these contexts may affect the game and be controlled within the game is the key to creating good games.

2.2.3 Rules in games

As seen in the section 2.2.2, rules are what actually make the playing experience possible to be shared with other players by shaping and limiting the magic circle. Since it is clear that games have rules and that rules are very important indeed, the question that next comes to mind is 'What kinds of rules are there?'

A good starting point for an answer is the typology of rules by Aki Järvinen, presented in his article 'Making and breaking games: a typology of rules' (Järvinen, 2003). He starts by noting that rules have no meaning as such, but only in the context of the game i.e. in connection with the elements of game. What then, are these elements? Järvinen (2003) distinguishes five building blocks that essentially make up a game:

1. *Components* are the objects that the players and the game-system manipulate during the game. Järvinen divides components into two classes: *player-objects* and *game-objects*. The first class consists of objects directly representing the player or players' status in the game. Examples of player-objects are player avatars, pieces

in *Chess* or maybe a card hand in poker. The second class consists of those objects which are not directly player-related such as non-player characters and decks in poker, for example. Components as such are static objects: they do not actually do anything. They have properties and qualities, which in the game's context assign meaning to them, but that's it.

2. *Procedures* both provide players and the game-system with the means to manipulate components and also define the structural relationships between components. These are the meat and bone of games, performable actions, something that makes change within the game possible. As components are static as such, procedures allow players and the game system to modify them: change their properties and the relations between components.

3. *Environments* are "the physical constraints of gameplay" as Järvinen puts it - they are the game boards, maps, playing fields and so on forth. Although typically games' physical environments have thematic information attached to them, like for example the board on the game *Monopoly* having each tile named as a street, it is important to distinguish these two things, environment and theme, from each other. Environments simply define the boundaries of accessible space to players, and

4. *themes* are the contextual settings of a game. They provide the 'story' behind the game and thus assign contextual meanings to actions, components and environments. A theme does not have to be strictly defined and it does not need to be narrative. For example, *Chess*' pieces convey a theme of a feudalistic kingdoms in battle with each other, but there isn't a story in *Chess*.

5. *Interfaces* are the players' means of access to the game. A typical interface is the graphical user interface in electronic games, but as Järvinen says, human body is also an interface in sports like football or ice-hockey.

Now, how does this relate to rules? Here is the key to understanding games: rules produce and govern these elements. What game designers actually do is that they create a set of rules to produce the game. Järvinen (2003) argues that there is a rule type for each element type: component rules, procedure rules, environment rules, theme rules and

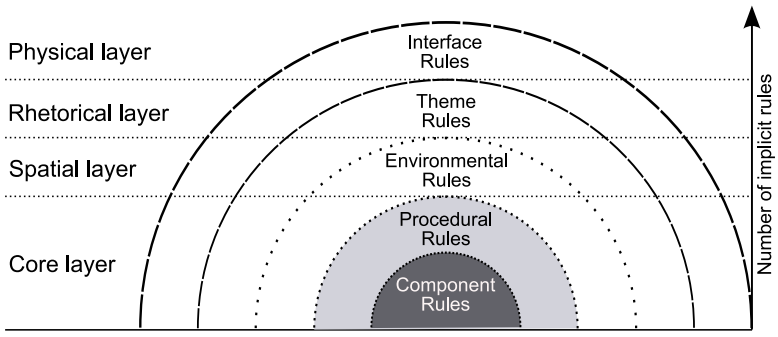

Figure 2.1: Järvinen's layers of rules (Järvinen, 2003, 77)

interface rules. Of these rules types, only the component and procedure rules are necessary for a game (Järvinen, 2003) as there can be no game without at least one component and some sort of a procedure. Saying this, he probably means that it is necessary for a game designer to create *explicit* rules at least for components and procedures - clearly there are rules that control at least the environment and interface. For example, in some dice games there isn't an explicit rule that states the physical boundaries of the playing environment, but players still can't go and throw their dice out of the window.

The most crucial point of Järvinen's article is his claim that these five rule categories can be seen as a stack of layers: core, spatial, rhetorical and physical layers, and that the complexity of representation and room for interpretation in rules increases as one moves up the stack. The core-layer consists of component and procedure rules, and is the lowest layer in the stack. On top of it is the spatial layer consisting of environment rules, and on top of it the rhetorical layer consisting of thematic rules. Finally, on the top of the stack is the physical layer made up of interface rules. (Järvinen, 2003) The point he is making is essentially this: the lower the layer of a rule is, the easier its effects are to control. The higher one goes on the stack, the less those rules are governed purely by the rules of the game. Thematic rules are a good example of this. Sometimes players tend to see thematic structures in very different ways and can act in manner that is not entirely foreseeable by the game designer. As a very crude and highly unlikely example, two players playing *Chess* could be too enamoured by the thematic elements in the game and choose only strategies that support their political views. One player being of working classes might do everything to protect her pawns, and the other as a royalist, might feed her pawns to the meatgrinder without a worry in the world.

Up until now, term 'explicit rule' has been used in this thesis to distinguish a written, beforehand thought out rule from a rule that clearly affects the game, but is something that the game's designer didn't either see as requiring written definition or just didn't realise its existence. A more thorough classification of rules is needed. Salen and Zimmerman (2004, 130)offer the following:

1. Constituative rules
2. Operational rules
3. Implicit rules

Operational rules are basically the same as the 'explicit rules' in this thesis - they are the written rules of a game providing players with guidelines as to how to play the game (Salen and Zimmerman, 2004, 132). The rule examples presented in section 2.2.1: '*Player with white pieces begins the game*' and '*Bishops moves diagonally*' are both operational rules.

Constituative rules are a bit harder to understand. As the name suggests, they are something that underlie a game, something that defines the basis of a game. For Salen and Zimmerman, constituative rules are the "abstract [...] sets of logical relationships that are not necessarily embodied in a material form or in a set of behavioral guidelines for the player" (Salen and Zimmerman, 2004, 132). By abstract, Salen and Zimmerman mean that constituative rules do not refer to the particulars of the game, i.e. they do not state colours of pieces or give instructions to the players as to how to move a piece. Instead, constituative rules define the logic of the game i.e. the formal structure of the game and they exist independently from players (Salen and Zimmerman, 2004, 132). In an example used by Salen and Zimmerman (2004, 128-129), the constituative rules of games *Tic-Tac-Toe* and *3-to-15* are the same mathematical logic of picking items from a set of nine and trying to create a special set of three items. In the case of *Tic-Tac-Toe*, the players select positions in a grid trying to get vertical, horizontal or diagonal rows. In the case of *3-to-15*, players choose from numbers 1-9 trying to get a set that equals 15. Constituative rules, like operational rules, have to be binding, explicitly and unambiguously defined rules (Salen and Zimmerman, 2004, 135).

If constituative and operational rules are the explicit and well-defined rules of games, then implicit rules cover the rest of the rules i.e. they are "the unwritten rules of a game"

(Salen and Zimmerman, 2004, 132). As the authors point out, the border between operational and implicit rules is very thin indeed: implicit rules can become operational rules in some situations - a stated or written implicit rule is an operational rule (Salen and Zimmerman, 2004, 134). It is a contractual matter: once the players agree to an implicit rule, it becomes a binding operational rule that all players need to follow.

This classification isn't meant to be a definitive typology of rules as there will always be rules that won't fit to any of the categories (Salen and Zimmerman, 2004, 134), but it does provide for a better understanding of rules. This classification fits nicely with Järvinen's typology: in each of his categories, rules can be either constituative, operational or implicit. Another important thing is that one can state his idea of rule interpretation opening up in upper layers using the concept of implicit rules: as one moves up in the layer stack, number of implicit rules increases.

Constituative and operational rules are important indeed as they define the game's formal structure, but implicit rules can be at least as important. Focusing on the idea that rules are explicit and unambiguous is asking for trouble. As Steve Sniderman puts it in his excellent essay "Unwritten rules":

> "The most powerful rules, the ones least likely to be violated, are those that are not stated explicitly, those that people have to infer or intuit. To state a rule is to invite players to break it, but to leave a rule unstated is to make its violation almost literally "unthinkable"" (Sniderman, 1999)

Implicit rules can have enormous effects in gameplay, so it is imperative for a game designer to try and recognize them as early as possible during the game design phase. Its about making educated guesses on how people behave in different situations, how they react to certain stimuli, and so on forth. Greg Costikyan lists a number of things that enhance gameplay in his essay "I have no words & I must design" and among these things are concepts such as diplomacy, roleplaying, socializing and narrative tension (Costikyan, 1994). These are clearly things that cannot be controlled solely by explicit and unambiguous rules. Naturally constituative and operational rules create the possibility for these actions, but it is very likely that it is the implicit rules that shape the forms these actions take.

2.2.4 Emergence and progression

"Every game is its rules, for they are what define it" (Parlett, 2005) is a nowadays famous quote by an English game historian David Parlett. It sums up quite nicely everything discussed earlier in this chapter. Rules define the game completely: a game is just a set of rules, nothing more. Rules define how a game is played, but interestingly enough, rules do not necessarily define how a game plays out or how good a game is. First of all, player experience cannot be bound by rules of the game. Like pointed out in the previous section, game designers can only make educated guesses as to how players react to different situations. Secondly, game rules usually produce *emergent properties*.

A common phenomenon in nature is that simple rules can create unpredictable complexity, like the functioning of an ant colony or as an ultimate example, the universe. This process is called emergence and the unpredicted behaviour, or phenomenon, the emergent property of a system. Games rely on emergence. Although it cannot be said that all games have emergent properties as there isn't even a satisfying definition of a game, emergence seems to be present even in the simplest of games. As an example, the childrens' game *Paper-Rock-Scissors* has just four core operational rules:

1. Paper beats Rock
2. Rock beats Scissors
3. Scissors beat Paper
4. Both players choose one of the components and both reveal their choices at the same time.

The game has clearly no inherent winning strategy that could be based solely on the rules since all the components are equally strong and each component beats only one of the other two components. Yet funnily enough, even in this game it is possible to achieve game mastery and start winning games more often than statistically probable. One way to do this is to study and learn the behaviour of the other player, like for instance "every time she twitches her eyebrow, she is going to use a rock". In other words, players form implicit rules of behaviour and strategy, which cannot be purported by

the game's constituative or operational rules. To take the unpredictable properties of *Paper-Rock-Scissors* even further, players have formed communities around the game and they even have world championships[2]. This is something that is simply unpredictable by just looking at the rules.

Emergence is a product of rules, there is no doubt about it. Rules create meaning within the game system by defining representational components and allowing actions with varying outcomes in a context of interpretation (Salen and Zimmerman, 2004, 368-372). Yet meaning itself is not sufficient for *meaningful play*. For a game to be meaningful, the outcomes of actions have to be both *discernable*, i.e. players need to know the consequences of their actions, and *intergrated* into the larger context of the game so that the consequences are not only restricted to the immediate outcome of the action, but also in the later stages of the game (Salen and Zimmerman, 2004, 24-25). The goal of game design is to produce meaningful play, so it is very important to understand how it can be done.

Jesper Juul (2005, 68) makes a distinction between two game types: games of progression and games of emergence. These two concepts are not exclusive of each other and only depict the two extremes in ways of creating meaningful play. For Juul (2005, 34), the discernability and intergration of outcomes are not yet enough for meaningful play, but in addition players need to be presented with *challenges* i.e. players need to invest effort into an action. The two types of games are simply distinctions between the means of presenting players with challenges: progression is the designed structure of "separate challenges presented serially" (Juul, 2005, 71) and emergence is the designed structure of presenting challenges indirectly (Juul, 2005, 68). Good examples of progressive games are the classic text adventure video games, such as *The Hobbit*[3], where gameplay progresses in a somewhat linear manner and each challenge usually has only one viable solution. Emergent games on the other hand present challenges in a more open manner: rules do not bind players to a small set of fixed solutions to the designed challenges in a game, but instead the openness of possibilities encourage players to explore new and possibly better ways of overcoming problems.

As a rule of thumb, emergence produces replayability and progression tends to lessen

[2]See World RPS Society (http://www.worldrps.com/), a club over 150 old and dedicated to the promotion of Paper-Rock-Scissors.

[3]Beam Software, 1982

it. This does not mean that progression is bad and emergence good, since there are clearly a lot of good games relying mostly on progression and still selling well. Likewise, there are a lot of bad games relying on emergence and not selling at all. Designing emergence or progression into a game, so that playing it becomes meaningful, is a hard task. Emergence by definition is unpredictable, and creating entertaining but aptly challenging problems is certainly not easy. A typical way to design a games is by means of trial and error: design a set of rules, see how it plays out and correct. This can be time-consuming, so a better way is to study other games and search for occurences of working examples of emergence or progressing challenges, and use them in their own games. As will be discussed in section 3.5, game design patterns can be of great value in this.

2.3 Constituents of meaning

2.3.1 Meaning and interaction

To understand what it is that makes up a game or, what it is that a game designer designs, one must look into the things that create meaning. Deducing from the discussion on emergence and progression in section 2.2.4 one can list the following: *representation*, *actions*, *information* as something to interpret and *goals* making up challenges. In addition to these at least two can be added: *conflict* and *social interaction*.

In order to discuss these notions more extensively, one more concept needs to be addressed. Reading books and articles on game design or listening to games researchers, one is almost bound to bump into the word *interaction*. Understood in its every day meaning, 'mutual or reciprocal action or influence' (The Collaborative International Dictionary of English), games certainly have interactive elements: players interact with each other and players interact with the game. The problem is that in the field of games research, the concept of interaction has grown almost out of context. So many definitions and meanings have been attached to it, that by now it is actually quite hard to understand what a person means by it. This is also the reason why its use has been kept to a minimum in this thesis. But, in order to understand the topics in this chapter, some description of interaction needs to be given.

Salen and Zimmerman, noting the same problem, suggest that instead of focusing on a definition, it should be more fruitful to see what forms interaction usually has in games (Salen and Zimmerman, 2004, 60). They begin by listing four ways, or modes as they call them, in which a person may interact with a game. These modes, as the writers point out, are not distinct categories but overlapping and often appering together in most games (Salen and Zimmerman, 2004, 59-60):

1. Cognitive interactivity is the way players interpret and react to the meanings within the game. This is a mode of interaction that could also be said to be present in books and movies as well.

2. Functional interactivity has to do with how players experience the game's material components. Using Järvinen's list of game elements from section 2.2.3, functional interactivity more or less defines the physical act of using game's interface.

3. Explicit interactivity is the designed interactivity of a system or "participation with designed choices and procedures". Examples given by Salen and Zimmerman are for instance clicking on the non-linear links of a hypertext-novel, following rules of a board game etc.

4. Beyond-the-object-interactivity is "interaction outside the experience of a single designed system" like fan cultures around games using elements from the game as raw material in their discussions.

Given this rather broad use of the term, it could be said that game design is mostly interaction design. Creating thematic structures and narrative elements for the game is about designing for the players' responses and interpretations. Similarly a game's interface is designed with players' physical capabilities in mind by making the interface as usable as possible. Designing gameplay is designing the explicit interactivity; the whole playability of the game often relies on well-designed choice-systems i.e. making the game such that players get satisfaction from making choices in the game.

The meaningfulness of the player's experience is dependent on the interaction between the player and the game. In other words, not taking into account the special interactive nature of games can lead to bad design. Yet, it is probably easier to understand games

in terms of the structures that actually produce meaning or at least greatly affect it, than it is to use a rather vague term such as *interaction*. The rest of the chapter will attempt to give short descriptions for one set of such structures.

2.3.2 Goals and actions

Actions without purposes would be pretty pointless. Rationality of an action is often measured by determining what can be accomplished by doing it.

Goals can be designed or not designed. Using *Chess* as an example once again, it has designed goals such as trying to check the other player and ultimately producing a checkmate to win the game. In the case of such goals, there are explicit and unambiguous rules defining them, but at the same time, *Chess* players have a number of goals that stem from, for example, the strategies chosen i.e. goals defined by implicit rules. A player on the offensive might reason that if she were to maneuver her bishop to a particular position, it would guarantee the removal of one particularly irritating piece protecting the opponent's king for example. This is a case of a subgoal: a stepping stone on the path to the ultimate goal of checkmating the other player.

Goals and actions are very closely linked to each other. Goals give meaning to actions by assigning values to the outcomes of the actions (Juul, 2005, 34): "Accomplishing these three things, I will be in a better position to reach this goal". One could say that goals justify actions by giving them reason.

Once again, looking at goals as something produced by rules, there can be at least three ways a designer can produce goals in games:

1. First of all, goals can be explicit by creating an operational rule to state so. Checkmate in *Chess* is a good example of an explicit goal.

2. Secondly, goals can be implied, in lack of a better word, in a way that rules either limit player behaviour or assign favourable qualities to some game components so that certain actions or strategies are clearly more desirable than others. As an example of an implied goal, getting to a highground position with a machine gun in game *Operation Flashpoint*[4] often gives a better chance of survival than

[4]Bohemia Interactive, 2001

running on an open field armed with a pistol. Another good example would be sneaking and stealthy movement in the game *Thief: The Dark Project*[5] - there is no rule that explicitly forbids moving in the open and making noise, but in the case of reaching one's goals, it wouldn't be a very good strategy.

3. Finally, goals can be the product of emergent properties like in the case of camping in many multiplayer first-person shooters such as *Action Quake*, a great *Quake 2*[6] -modification, or *Counter Strike*[7], a *Half-Life*-modification. Camping is seen in these games as a despicable act, where a player hides in some secluded corner of the map and simply waits for the unsuspecting players to prey on. This was something that very likely was never intended by the designers of the game, but it become such a good strategy and most players deemed it as destroying the playing experience, that measures had to be taken to forbid or restrict this behaviour.

Understanding how goals can be used to affect player behaviour is crucial for a game designer. In addition to understanding goals as something players attempt to achieve, such as for instance evading something, collecting a set or destroying something, one should also understand the ways goals can be related to other elements in the game. Björk and Holopainen call these relations *goal structures*, which are "closely related to goal patterns but are not goal patterns in themselves" (Björk and Holopainen, 2005a, 309). Creating structures around and of goals can be used to alter the meanings otherwise purported by the goals. For instance, consider a game of ice-hockey. Although normally the goal is to win the game, in a tournament it may actually sometimes be more meaningful to lose the game in order to get a more suitable opponent for the next game. In this case the tournament creates a goal structure, which effects the way individual goal of winning a game are perceived by the players.

Björk and Holopainen list 17 types of goal structures which can be created in essentially three ways (Björk and Holopainen, 2005a, 309-338):

1. Modifying the characteristic of the goals in question by, for example, making some goals unknown to players or changing some goals to be interferable by other players.

[5] Looking Glass Studios, 1998
[6] id Software, 1997
[7] Valve Software, 1998

2. Modifying relations between multiple goals as in the case of tournaments where the goal of winning a single game can be more meaningful in the context of winning the next game.

3. Modifying relations between players and goals making for instance some goals mutual to a group of players where upon it may be reasonable for players to ally themselves to better pursue the goal.

Goals essentially define a reason to do an action, but they do not define the action itself nor do they make the action something worthwhile doing. For that one needs conflict.

2.3.3 Opposition and conflict

An often repeated quote by Greg Costikyan of games without opposition being variants of "let's all throw a ball around" (Costikyan, 1994) has some truthfulness in it - games without opposition or conflict are generally quite dull. If a game does not offer one any kind of challenge in terms of trying to beat someone or something, then it can be questioned whether such a game is a game at all. However, conflict does not automatically mean that the opposition should always be some other players, instead it can be the game system itself or even simply the player herself.

Conflict itself isn't necessarily something as dramatic as battle or a clash between players, but its core is more in making players struggle to achieve some goal in the game, as pointed out by Salen and Zimmerman (2004, 250). Generalised, one could say that if goals are what give actions their reason, then conflict is what gives actions their meaning by making them something that is worthwhile or fun to do.

2.3.4 Information

A very important element in creating meaning and meaningful play in games, *information* has various definitions. In a broader and often used sense, information is processed data which is somehow meaningful to its interpreter (Floridi, 2005). Understood thus, information is certainly present in all games: players turn the raw data they have on the game into meaningful information which they use for making decisions during the

course of the game. For example, in *Chess* the composition of the pieces on the board is the raw data which needs to be processed and turned into strategic information on the game: what is the player attempting and how could one best exploit the situation.

In a more narrow and stricter sense, information is a unit of measure. At first sight, information in this sense seems almost the opposite of the more broader definition: in information theory, information has nothing to do with meaning, instead it measures the amount of uncertainty in a message (Shannon, 1948). This definition provides an interesting standpoint to study how information creates meaning in games. A message that is totally predictable, i.e. its recipient is certain that the message is correct, contains very little information as opposed to a message that could contain anything, which in turn would have a high degree of information (Salen and Zimmerman, 2004, 193). As Salen and Zimmerman put it, one could also say it in other words: information measures the freedom in decision making - the more freedom to choose, the more information is present (Salen and Zimmerman, 2004, 194).

How then does this relate to games? Well, it opens up a possibility to analyse games as information systems using the terminology and theories developed in the field of information and communication theory. Salen and Zimmerman (2004, 196-198) only touch the surface of this, but even so, give interesting examples of how these notions could be applied in games. Quite unlike in communication theory, games usually rely on balanced amount of information. Information increases the number of choices a player has and also induces guessing in situations where a player tries to interpret information on other players' actions, so a game with no or very little information would quite likely be very dull game.

As examples on how to use concepts from information theory in games, Salen and Zimmerman (2004, 196-198)point to a common children's game *Telephone* and crossword puzzles. *Telephone* is a game where a message is passed by players whispering it to another and the fun arises from changes in the original message. In information theory's terms, the game relies on the addition of *noise* in the channel (Salen and Zimmerman, 2004, 196). Crossword puzzles on the other hand rely on *redundancy* in the message (Salen and Zimmerman, 2004, 198). Redundancy in communications systems refers to how much of the message needs to be transmitted in order for the message to still be understandable - even if some of the message is lost, the rest can sometimes

be understood because of redundant patterns in language. A person filling crossword puzzles is doing just this, trying to fill out as much of a word as one needs before the rest of the word can be guessed by the redundancy in the word. The game *Hangman* uses the same mechanism.

Although the information theoretic view on information is indeed very interesting in terms of game design, the notion of information in its broader sense can be used quite effectively for creating meaning in the game. Björk and Holopainen make a distinction which is quite useful for speaking about information: they distinguish players' experiences and strategies from the *game's state*, which in turn is the "collection of all values of all game elements and the relationships between them" (Björk and Holopainen, 2005a, 8). A game designer can directly control only the amount of information players have on the game's state by defining rules for information exposure in different states of the game. This could mean giving players all the information on the game's state to the players, as in *Chess,* or it could be something more limited or it could even be something totally falsified, but it must be something: players need at least some information on the game's state, otherwise the game cannot be played.

Yet, although not directly, a game designer can also affect players' information on other players' strategies and experiences, for instance, by allowing players the means and incentive to communicate information on their own situations to other players. For example, in *Diplomacy*, the current state of the game is revealed to all players i.e. everyone knows how many armies and fleets other players have and where those pieces are located. What the players do not know is how the other players are going to move their pieces in the next turn. As virtually all moves after the first few turns require support from other players and as there is no built-in mechanism to create binding alliances between the players, the game turns into an information struggle where players try to convince other players on their intentions and at the same time try to decide who to trust or not. In the basic version of the game where all players play in the same room, players have access to such information as past moves, possible moves in the next turns and, most importantly, how other players behave i.e. who do they talk with and how they seem.

Game's interface is the most important source of information for the players in the game. Players make their choices based on this information and a very typical winning

strategy in some games is to accurately interpret the information regarding other players actions or state of mind like in *Poker*, where the effect of chance is reduced by giving as little clue as possible on one's own situation but also by trying to correctly assess the situations of other players. So, generally speaking, one might say that information serves for two uses in games: it provides a basis for players' actions, but at the same time players can also use it to influence other players' actions.

2.3.5 Representation and narratives

An important tool for creating meaningful experiences in games is certainly the use of stories and themes, but in order to understand them, one must understand games as representational spaces. Games are filled with elements depicting various different kinds of things, which the players interpret in some way i.e. create meaning out them (Salen and Zimmerman, 2004, 364). For instance, the *Chess* pieces in the wide-spread version of the game are all depictions, or more precisely, representations of social roles in a feudal or monarchic society. Although the physical form of the *Chess* pieces can vary a lot from set to set, players typically know which piece is which due to the generally used representations such as crowns on kings and queens, the hat on bishops, and the sizes of the pieces depicting their importance, for example.

Clearly, representation in games is not only limited to objects and elements but actions too are representations, just as for instance in theatrical plays. As a very simple example, the way knight moves in *Chess* conveys the idea of horse jumping over other pieces. In first-person shooters such as *Quake*[8] or *Half-life*[9] and especially in massively multiplayer online role-playing games such as *World of Warcraft*[10], representations of actions are naturally a very important sources of meaning for the players, conveying things such as threat posed to the players and so on.

A feature that really sets representation apart in games is the concept of metacommunication. The safety of the magic circle and the fact that players know they are playing a game, allows them to view representations of actions and other objects within the game in a different manner than they would outside the game (Salen and Zimmerman,

[8] id Software, 1996
[9] Valve Software, 1998
[10] Blizzard Entertainment, 2004

2004, 371). This allows designers for instance to create fun gameplay in contexts that would probably be morally unacceptable or otherwise impossible for most players in real world.

There are many games which are heavily story-driven and sometimes seem more like a movie than a game, but as pointed out by Henry Jenkins, narratives in games are too often reduced to thinking them in terms of simple story-telling (Jenkins, 2004). Clearly not all games have an explicit story behind them and some games, like *Checkers*, even seem to lack a visible theme, but it does not mean that these games are not narrative. Using a definition from a literary theorist, J. Hillis Miller, Salen and Zimmerman (2004, 380-381) define narratives as constituting of the following components:

- *Situation*, with the narrative progressing as events, with sequences leading to a change of event.
- *Character*, meaning that the narrative is presented through a system of representation.
- *Form*, where the representation is constituted by repetition and patterning.

Understood this way, all games are narratives. For instance, *Checkers*, has an initial state and progresses over time with players taking turns to change the situation, its pieces, *men* and *kings*, are presented through a system of representation, and there are plenty of repetition and patterning in the game. It may be that most people do not play *Checkers* because of the compelling narrative it offers, but the point is that the way games are structured allows for easy use of narratives as sources of meaning for the players.

Narratives offer compelling tools for designers. They can be used to shape the gameplay, guide players, create tension, immerse players and so on forth. Although the topic of how to use narratives in creating meaning is simply too large for this work, in terms of the rule-like principles affecting the game rules and the players, literary theory offers a great set of guidelines which can be used for game design.

2.3.6 Social interaction

Directly usable only in multiplayer games, social interaction is a great way of creating meaning within the game. Although the majority of the current computer games seem

to be single-player, games as such are usually social by nature and in similar fashion, the players of single-player games also often create a social element around the game by creating community websites and other social medias.

Salen and Zimmerman give a lengthy discussion on the subject in their book Rules of Play. They note that games as a bounded play community are actually a form a social contract between the players. This contract has two key properties which make the social play a very powerful element in games:

1. The meanings and values within the game are created by the players through play.
2. The contract makes sure that the space defined by the game is a safe one allowing for more possibilities for actions for the players. (Salen and Zimmerman, 2004, 470-473)

All the social interaction created by games is certainly not controlled by a social contract. There are plenty of social activities that take place in the meta-level such as the previously mentioned player communities and then there are also the border-case participants of some pervasive games: people who do not even know they are participating in a game.

The meta-level social play is certainly important and interesting, but it is not necessarily something the designer can directly control, nor is it as interesting as the safe play within the social contract. Social play in games is quite heavily connected to the concept of conflict. Although the social aspects of games at large are definitely beyond the scope of this work, it should be stressed that out of the struggle comes one special feature of games and play: the safe competition between the players.

Designers have quite a lot of tools for creating meaning by social aspects of the game. Björk and Holopainen (2005a, 237-276) define 30 game design patterns that can be used for controlling social interaction. They make a note that there is a difference between the normal social interaction between the players, like for instance chatting during a game of *Chess*, and the stimulated social interaction, which in turn is a product of the game rules (Björk and Holopainen, 2005a, 259). Designers can certainly affect the normal social interaction in many ways, such as simply by providing a channel for chat-

ting or by making sure that players are in close physical contact and so on forth, but the ways of creating stimulated social interaction can probably be used more controllably.

A good example of stimulated social interaction is bluffing in game *Diplomacy*, where the whole game revolves around players trying to bluff and others trying to find out whether someone is bluffing (Björk and Holopainen, 2005a, 269-271). As the best means of doing so is to talk with other players and try to read their body language, the result is really stimulated social interaction.

3 Methodological Game Design

3.1 Introduction

The notions of structure and the principles of game design developed and examined in the previous chapter form the theoretical basis for this chapter.

A game design method has two important requirements: it should provide a structure and rules or guidelines for creating relations between the structural elements. As has already been mentioned, the meaning-producing elements form the structural part and game design patterns, which will be discussed and explained in more detail at the end of this chapter, form the rules for describing and connecting the elements. Although game design patterns can be understood also as a method, here they are seen as principles of game design capable of forming the rule-structure for virtually any method thus forming a sort of an specification language for game designs.a

In section 3.4, a number methodological approaches to game design are introduced.

3.2 Defining the Game Design Method

> "Game design is the process by which a game designer creates a game, to be encountered by a player, from which meaningful play emerges." (Salen and Zimmerman, 2004, 80)

Game design is essentially a process of *rule design*. It is about creating rules that work well together and create meaningful experiences for players. At the same time it is about minimising the effects of rules that are not part of the actual game rules, yet effect the game in one way or another.

One might say that the game design phase is the most important phase for the whole game development process. The decisions made during this phase are in fact more

or less detailed instructions and restrictions for the actual implementation phases for the game. In a real-world project, the implementation is typically a very costly and time-consuming phase in game development, so the ideal result for game design is to describe the game as completely as possible for the implementors.

Game development could certainly benefit from using methodological approaches to game design but unfortunately, game design research has been receiving quite little attention in the past. Although the interest in researching game design is clearly stirring, there are still a large number of important basic definitions and research questions lacking thorough studying. Although these issues are mostly outside the scope of this work, one particular problem needs to be addressed: what exactly is a game design method?

Bernd Kreimeier (2003a) lists four desirable qualities:

1. A method should be related to game design i.e. the focus should be on the gameplay and the structures supporting it, not marketing or production issues.
2. There should be procedural instructions on using the method.
3. It has to be abstract enough so that the method is applicable to a larger number of situations. A method that gives instructions on how to design the same game over and over again is not very useful.
4. And finally, a method should have some formal structure and organisation to allow for easier use of the method but also to make users focus on the correct things.

Kreimeier's list defines the general structure of a game design method but there are at least two important questions that need to be addressed:

1. Why should one use a design method?
2. How does one decide what method to use?

Kreimeier's answer to the first question is that as tools to "extract, identify, refine, and organize knowledge [...] [Methods] should encourage introspection and observation,

and turn reactive choice into conscious, proactive planning decisions." (Kreimeier, 2003a) This is clearly the most important benefit of using a design method, the ability to make justified decisions and especially to be able to at least roughly predict their consequences to the whole game.

The second question i.e. the problem of evaluating design methods is a quite complex one since the goodness of a method clearly comes down to individual preferences. For some, a good method is one that produces games that sell best, for some it can be the quality of games, some need a method that produces simple games and so on forth. There are so many different evaluation criteria applicable to games and game design that it is hard to imagine a method suitable for all game development situations.

By extending Kreimeier's definition a bit, one could say that a game design method has two core properties which are of great importance to a game designer:

- it helps to structure the game design early on in the game development process, and
- it allows the designer to create meaningful experiences according to already tested principles of game design.

The granularity of these two attributes, i.e. how detailed is the structure and how specific are the principles, can vary from method to method. As a rule of thumb, it could be said that the finer the granularity of either property, the better the method for controlling the decision making process, but at the same time it gets harder and more time consuming to use.

As to what kinds of methods or approaches there already are in game development business, Bateman and Boon (2005, 5-8) list seven commonly used categories:

1. *First Principles* -approach where the design begins by deciding on the goals for the whole process and then designing a game to fit these principles.
2. *Clone and Tweak*, where one takes an existing game as the basis for the design and then applies some modifications and transformations to it in order to create a new game.

3. *Meta-Rules* as a formalised approach to game design, defining rules to govern the design process. Game Design Patterns Björk and Holopainen (2005a) and the 400 Project Falstein and Barwood (2003) are good examples of this.

4. *Expressing Technology*, in which some new technology dictates the requirements for the game i.e. a game is designed to express the technology.

5. *The Frankenstein Approach*, where the design starts by utilising already available materials such as artwork or software and see what they can be turned into.

6. *Story-Driven Design*, where the design revolves around and is made to fit a storyline.

7. *Iterative Design*, in which the design is revised and repeated until the result is satisfactory.

In terms of Kreimeier's definition, the above approaches are not methods but only categories under which the prospective methods could be placed. In fact, excluding the Meta-Rules, these approaches are probably better described as defining the practical aspects of game design or more precisely, how to start the process. Although all approaches shape and guide the design process differently, it is hard to imagine that the rules for creating gameplay experiences or the formal structures of games vary from one approach to another.

The methods and tools introduced in this chapter represent some of the more formalised examples of *gameplay* design aids[1]. Each defines either the formal structure or some formal rules for creating game designs: MDA, which is discussed in section 3.4.2, and the Player-Centred Design, which is discussed in section 3.4.6, focus on designing the *experience* of the players while the others have a more abstract structure of *game elements* as the starting point for the design.

It should be noted that there are also other formalised approaches which could have been included in this work such as the ones by Chris Crawford (2003), Greg Costikyan

[1]There is an ongoing discussion within the games researcher community about the core aspect of games, where the ludologists focus on the gameplay mechanics and the narratologists emphasising the importance of a good storyline (Jenkins, 2004). Game design patterns are clearly gameplay mechanics oriented.

(1994), Richard Rouse III (2001) and maybe also the one by Rollings and Adams (2003). There is also the very interesting *Game Ontology Project* (Zagal et al., 2005), which is developing a framework for analysing and understanding games, but it would require too extensive coverage for it to fit this thesis. The fact that these aren't discussed in this chapter is then not due to their lack of merits, but mainly due to space and scope issues.

3.3 General description of game design process

The design phase of a game consists roughly of the following three separate stages (Fullerton et al., 2004, 140-181):

1. *Idea generation*, in which the idea of the game is first created. The result from this phase is usually a crude outline of the game maybe describing the theme and some player action or goal.

2. *Conceptualisation*, in which the idea is refined into a concept defining the game more accurately stating for instance target groups, simple storyline, core mechanics etc.

3. *Prototyping*, in which the concept is tested by building a simple prototype of the game. The prototype itself can be anything from a sheets of papers up to a computer program, but it has to be quick enough to build and functional enough for the testers to be able to test the mechanics of the game.

Normally the process is also *iterative* so that if the result from one stage is not satisfactory, designers can go back to an earlier phase and work on the problems there before moving on. The end result of this process is the design document[2], which will be the basis for implementation. The extent and specificity of the design document can vary greatly according to the method used, but typically it contains elements such as the description of the gameplay, possible storyline and characters and so on forth. The

[2]Kreimeier (2003a) actually lists the game design document as a method in its own, but since virtually all professional game development relies on the design documentation, it makes more sense to view it as an essential part of all game design.

purpose of the design document is to communicate the game concept to those implementing the design but also to the funders as well (Fullerton et al., 2004, 372).

The game design phase itself is not a discrete step before the other parts of the development process such as software design and implementation. The requirements specification can be started already in or at least right after the conceptualisation phase. As outlined by Walton (1998), requirements specification at this point should already provide a requirements baseline for the whole project i.e. take into account all departments involved in the process such as for instance marketing and support in addition to creative and technical aspects. The design document in turn will provide the basis for the requirement analysis, system architecture design and the detailed module specifications which finally lead into the implementationWalton (1998).

Quite interestingly, most of the methods and tools discussed in this chapter deal intensely only with stages 2 and 3. Idea generating is viewed more or less as given or at least something that all designers can decide for themselves. This is of course quite understandable since ideas do not usually represent many of the elements that are often attributed to games but are more or less the raw source material which can be molded into games by using the formal structure and rules of a method.

3.4 Game design tools and methods

3.4.1 A model to support the design of multiplayer games (Nussbaum et al.)

In their article "*A Model to Support the Design of Multiplayer games*" (Zagal et al., 2000) Zagal, Nussbaum and Ross lay out a relatively simple method for the initial steps of multiplayer game design. Although primarily intended for multiplayer games, this model can also be used in the development of other types of games.

Before going further with the description of the model one must understand what the authors mean by the game's concept. In their view it can be described by the following groups of elements:

1. *Rules and goals*. Rules seem to refer to operational rules, but other than that, these are essentially the same as discussed earlier in the sections 2.2.3 and 2.3.2.

2. *Props and tools*. These are the components of the game. Props are used for purely decorative purposes i.e. they can be used to refine the representative elements of the game. Tools, on the other hand, are something that players can use in order to play the game.

The method purported by the article is, not so much a procedure, but more so a formal description of the characteristics of multiplayer games in relation to the aforementioned elements, that a designer should focus on when defining the concept paper. The characteristics and their relations are depicted in Figure 3.1.

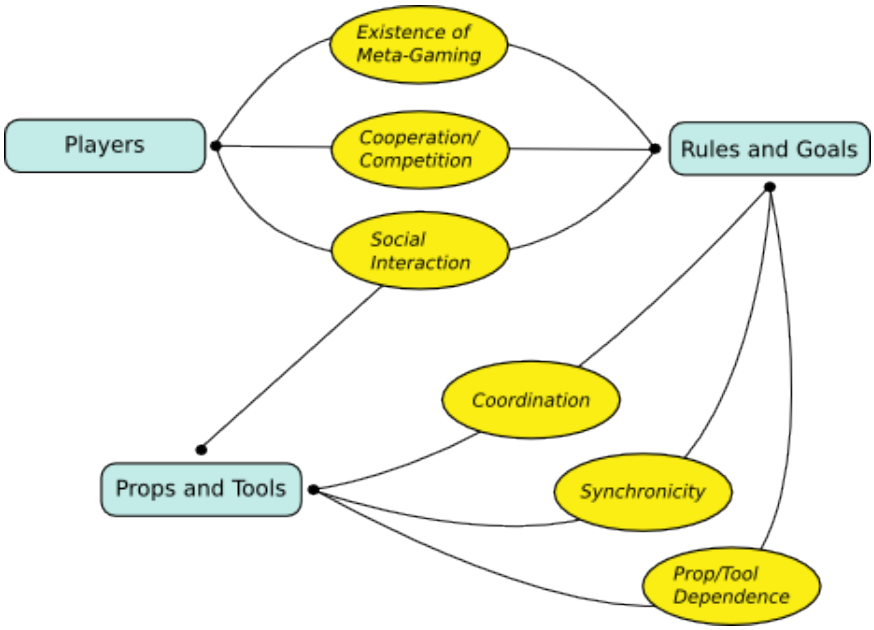

Figure 3.1: Multiplayer game characteristic in relation to game elements

Social interaction as well as *Competition and Cooperation* have already been discussed at length in the previous chapter, but the other characteristics require some explaining.

Synchronicity is the temporal advancement of gameplay in regards to players' actions. It is divided into three distinct types of synchronicity: *concurrent*, *synchronous* and *asynchronous*. In concurrent games, players play the game simultaneously and each player can act at will. Typical examples of such games are for instance most multiplayer versions of first-person shooters like *Counter-Strike*[3] and *Return to Castle Wolfenstein: Enemy*

[3]Originally a *Half-Life* -modification.

Territory[4]. In synchronous games, players play simultaneously, but take turns in acting. As an example, *Chess* is a synchronous game. Finally, in asynchronous games, players can attend the game instance (more or less) at will without being dependent on other players' concurrent participation. As an example, most Play-by-mail -games, such as *correspondence Chess*, are asynchronous by nature, allowing players to decide when to submit their moves, for example.

Coordination is about who controls the flow of the game process. A game can be controlled by one person, like referee in *Soccer*, or the control can be given to more or even all people participating the game like in the case of *Hide and Seek*. By *Prop and Tool Dependence*, the authors refer to the relation between the game's operational rules and the components of the game. In case one could swap all the components into something else, then the game would not be prop and tool dependent at all.

Using this model as a design method involves making sure that all the characteristics are satisfiably defined. The model guides this process by indentifying dependencies between the characteristics and thus points the designer to see the consequences of her decisions.

3.4.2 Mechanics-Dynamics-Aesthetics

A very interesting look on game design is the MDA (Mechanics-Dynamics-Aesthetics) -approach by Mark LeBlanc(Hunicke et al., 2004). MDA is not so much a method in the traditional sense, but more a framework for the game design process. It could be labeled as *experience-design*, in which the focus and the starting point of the design process are the experiences of the future users of the product. For LeBlanc, these experiences are certain kinds of *fun*. Selecting them in the beginning of the process defines the design space for the game making future design decisions subordinate to these. In other words, the kinds of fun are the *emotional design requirements* for the game design process (Hunicke et al., 2004).

Mechanics, dynamics and aesthetics are the building blocks of the theory. The mechanics are the formal basis for the game i.e. in terms of the rules structure discussed in 2.2.3 mechanics could be defined as the constituative and operational rules of the

[4]Splash Damage, 2003

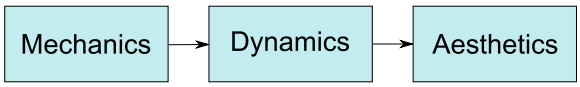

Figure 3.2: MDA

game. Dynamics is the gameplay produced by the mechanics, or as LeBlanc puts it, the run-time process of a game. Finally, aesthetics are the players' emotional responses to the dynamic properties of the game - in other words, the 'fun'.

In the article MDA: A Formal Approach to Game Design and Game Research the authors provide a taxonomy of eight kinds of fun (Hunicke et al., 2004):

1. Sensation, game as sense-pleasure.
2. Fantasy, game as make-believe.
3. Narrative, game as drama.
4. Challenge, game as obstacle course.
5. Fellowship, game as social framework.
6. Discovery, game as uncharted territory.
7. Expression, game as self-discovery.
8. Submission, game as pastime.

A player of a game can experience lots of different kinds of fun in terms of the above list, even the kinds that weren't specifically designed to the game. In an example from the same article, *Quake* is defined as a game of *Challenge*, *Sensation*, *Competition*[5] and *Fantasy* (Hunicke et al., 2004), yet to some players the funness of the game might be the social aspects of it: playing with your friends, discussing the game with them and so on forth. Similarly, although mostly a game of *Challenge*, for some players *Chess* could also be a game of *Fantasy*. The player experience cannot be *fully defined* with just an analysis of the game, but it can be *anticipated* by producing the right kinds of dynamics, which in turn can be anticipated by using the right kinds of mechanics.

[5]Competition is used throughout the article as a kind of fun, but not included in the list.

From the game designer's point of view, aesthetics are the goals, or the design requirements, for the whole design process. One chooses the kinds of fun the game should create and then starts to formulate a set of rules that could produce it. Although MDA does not fully define these rules, LeBlanc's article *Tools for Creating Dramatic Game Dynamics* (LeBlanc, 2005) is clearly a beginning in this direction, as he introduces a number of dynamics giving rise to narrative fun. The mechanics are all general enough to be usable in any game. One example of these mechanics is *escalation*, which creates dramatic tension by raising the value of some game resource as time passes. This could be for instance the number points given for successful actions so that as the game progresses towards the end, the player can get more and more points for his actions. This keeps up the dramatic arc by making sure that the game cannot be lost in the beginning of the game, and also heightens the tension towards the end, where the bigger prizes await.

3.4.3 The 400 Project

The 400 Project, started and lead by Hal Barwood and Noah Falstein (2003), is a strictly rule-based approach aiming to build a list of practical rules for game design. The goal of the project is to have a list of rules that can be applied immediately to and also improve the design of a particular game. The rules are not formalised components that can be used to constructing a game from the ground up nor are they meant as strict imperatives but more as useful guidelines helping designers to identify possible flaws and to fix them in an already existing design.

The rule names are meant to be concise and easy to remember, e.g. *Provide Clear Short-Term Goals* and *Emphasize Exploration and Discovery*. Although the set of rules do not have a strictly defined formal structure, they can have loosely defined relations to each other. The format for describing a rule consists of five parts:

1. An imperative statement of the rule, which also acts as a name for the rule.
2. A description of the domain of the rule. Examples of domain are *psychological*, *meta* and *basic*.
3. Rules or circumstances which take precedence over the rule.

4. Rules or circumstances that the rule takes precedence over.

5. Examples and counter-examples from well-known published games.

The master list for the project currently has 112 rules although, as a work in progress, there are rules with missing and incomplete information. The relations to other rules are not included with the list.

The rules in the project cannot be defined as a game design method, but they are a tool to aid the design process especially in the stages of conceptualisation and prototyping.

3.4.4 Game Design Workshop by Fullerton, Swain and Hoffman

A whole book on organising workshops and general advise on how to become a game designer, Game Design Workshop by Fullerton, Swain and Hoffman (2004)is not exactly a description of a method, but it does outline an approach to game design that is worth reviewing. Interestingly they also pay attention to the earliest stages of design, the ideas, listing different kinds of idea generation techniques. Although they stress the importance of iterative idea generation, they do not unfortunately define any rules for making decisions already in this phase and therefore leave the choosing and formulation of the ideas solely to the designer.

The formal structure of game designs is defined as consisting of eight components (Fullerton et al., 2004):

- *Players*, including the number of players, roles of the players and also the interaction patterns for the players. Examples of interaction patterns are cooperative, single player and team play.

- *Objectives*, for which the term *goals* is used in this thesis. The authors give examples of objectives such as capture, alignment and rescue.

- *Procedures* are both the actions players can take but also the system procedures, which are the game system's responses to player actions.

- *Rules*, by which the authors mean the operational rules of the game i.e. rules that guide and control the game elements and player actions.

- *Resources*, which are those game elements that have both utility but also scarcity within the game. Typical game resources are the players' lives, currency and actions.

- *Conflict* is the emergent property which is produced when players try to reach their objectives. The authors list different sources for conflict such as obstacles, opponents and dilemmas.

- *Boundaries* simply define what belongs to the game and what does not. They can be physical as in the case of *Soccer*'s borders for instance, but they can also be temporal i.e. when to play, or social i.e. who is playing.

- *Outcomes* are quantifiable and measurable ways of assessing the course of actions players take during the game. In most games this will mean that players can be divided into winners and losers, but there are also games that do not have a winning condition, in which case the players assess the outcomes of their actions using some other measures.

The methodological part of this approach is quite simply a guideline to just pay attention to the various parts of the formal elements in games. The authors offer quite extensive coverage of each element with examples and questions to guide the decision making process.

3.4.5 GameGame

A fun method for creating ideas or game concepts is Aki Järvinen's *GameGame*[6] in which the method is actually a game, where the goal of players is to design a game. The formal structure is based on the article discussed earlier in 2.2.3 although Järvinen has developed his theory further in his yet unfinished dissertation[7]. The formalisation consist of seven elements, which also serve as different types of cards - i.e. game components - in the *GameGame*. *Components*, *Environments*, *Themes* and *Interfaces* are the same as in 2.2.3. In the game they are just blank cards defining structure for the game

[6] http://gamegame.blogs.com/
[7] http://www.gameswithouthfrontiers.net/

to be designed - it is the player's job to give them further meaning. In addition to these four, *GameGame* also has *Goals* which like the former ones are blank and left for the player's to define, *Mechanics* which are collections of procedural rules defining actions to be used in the game, and finally *End & Victory Conditions*, which are the outcome rules for the game.

As a game design method, *GameGame* is a very interesting mix of formal structure and fun gameplay. It can be really useful in allowing players - designers - to focus on utilising their creativity and at the same time create formally structured game concepts.

3.4.6 Player-Centred Design

An approach by Ermi and Mäyrä (2005) has a base rooted in the user-centred approaches commonly used in software development to test and design the usability of programs, but as the authors stress, Player-Centred Design is still quite different. Instead of focusing on the *users*, which has the connotation of viewing software, or in this case, games, as a tool, Ermi and Mäyrä focus on *players* and speak of designing entertainment experiences. Although it is hard to categorise Player-Centred Design as a method in terms of the definition discussed earlier in section 3.2, it is a very interesting meta-method which helps designers to refine their design principles in a per-concept -basis.

At the heart of the approach is a scenario-based player study method in which the prospective game ideas or concepts are turned into gameplay scenarios, which in this case were realised in form of a comic strip. Then a group of test participants are asked to fill out a questionnaire concerning the respondents' playing habits and other background information. Next a smaller group of respondent are asked to participate in interviews in which the comic strips are finally presented to the participants and they are asked to evaluate their emotional responses to the scenarios. After the individual interviews, the scenarios are also discussed in depth in a group discussion, after which the all the participants fill out a final questionnaire concerning more specific details of the game concepts.

The results from the player study are then turned into game design requirements such as ”Allow different modes of gameplay and support various player types” or ”Support

the players' control and feeling of control regarding lottery gaming" thereby allowing the feedback from the players guide the decision making process already in the conceptualisation stage and onwards. The authors point out that using this approach is a time-consuming and difficult effort, but it does have the benefit of producing research knowledge that can be reused in future projects.

3.5 Game design patterns

3.5.1 Related approaches

In 1999 Doug Church published a paper called Formal Abstract Design Tools (Church, 1999), in which he advocated for the need of a common language of game design using a vocabulary based on dissecting games and analysing why they work or do not. The end result of this analysis would be a set of formal rules, FADTs, that would give designers indication as to how varying game mechanics work together and also, would provide designers with means to discuss games in more accurate terms. The focus of his approach was to formalise and abstract game mechanics beyond the scope of a specific game instance, so that the language would describe games in general instead of categorising them to different game genres for instance.

Church's paper has stemmed a number of papers describing similar approaches such as *The 400 Project* by Noah Falstein and Hal Barwood (2003), *A Case For Game Design Patterns* by Bernd Kreimeier (2003b) and most recently *Towards an Ontological Language for Game Analysis* by Zagal et al (2005). Although each paper calls for the same thing as Church: create 1) a collection of tools to help and guide game designers, and 2) a common language for game design, they all vary in depth of analysis and all take a somewhat different viewpoint in the matter.

The 400 Project has already been discussed in section 3.4.3. The Game Ontology Project by Zagal et al. on the other hand, is less of a rule-based approach. It aims to identifying and describing the structural elements and the interrelations between them so as to provide for a framework of describing and analysing games instead of giving instructions for designing games. These are both very interesting and promising projects, but for the purposes of this thesis they are both too limited.

3.5.2 Characteristics of game design patterns

Game design patterns offer the best of both worlds: they describe the interrelating structures in enough detail to be useful to game analysis, but they also give solid instructions as how to design better games. Game design patterns have their roots in architectural design patterns first introduced in the book A Pattern Language: Towns, Buildings, Construction by Alexander et al. (Alexander, 1979) For Alexander, a pattern describes some common problem in architecture and then provides a proven solution to it. This is also they way Kreimeier (2003b) sees patterns in his article, but later on, through the work of Holopainen and Björk, game design patterns evolved into something a bit different.

The view of game design patterns being essentially problem solving tools, was rejected. The authors argued that, game design patterns should be more useful as tools to support the creative process of design instead of being just means to removing parts from an already created design. Secondly, patterns consist of a quite a large number of interrelations so that each modification could result in multiplicative effects in parts of the design that originally were not seen as a problem at all. (Björk and Holopainen, 2005a, 34)

The idea behind game design patterns is the notion of seeing games can "as an entity put together by a number of smaller components" (Björk et al., 2003). As the quote hints, a component is something delimited and identifiable, a specific aspect of gameplay. A very apt way of describing architectural and other physical constructs, but for games, not so prudent at first sight. A closer comparison might be made by looking into object-oriented programming, where a non-physical entity, computer program, is described as a collection of interrelated objects. Yet it isn't a very good comparison after all. If games consisted only of a number of explicit and unambiguous rules defining the game's formal structure, it would be fine, but games are more than that. There are also the players' experiences, implicit rules and emergence. As one cannot describe an aspect of *gameplay* without referring to these three elements of games, the possibility of formal and abstract definition must be ruled out.

Holopainen and Björk describe their patterns as semiformal, meaning that although the pattern descriptions themselves are non-formal, the whole of the collection has formal

structure due to the relations between patterns (Björk and Holopainen, 2005a, 35). Each pattern can be related to other patterns in five different ways: a pattern can *instantiate* or *be instantiated by* another pattern or patterns, a pattern can *modulate* or *be modulated by* another pattern or patterns, and finally, some patterns can be mutually *potentially conflicting*. These relations require a bit more explaining (Björk and Holopainen, 2005a, 35-36):

- Instantiation describes a situation where the existence of a pattern or patterns may automatically guarantee the existence of some other pattern.
- Modulation defines the relation where an existing pattern or patterns can be used to modify the qualities of another existing pattern.
- Potentially conflicting patterns can sometimes, but not always, render each other impossible to coexist.

There are some qualitites of this relational formalism, that must be noted. Firstly, it must be stressed that the relations between patterns only describe *structural potentiality*. As an example, given any pattern A, there is currently no way to deduce from the structural information whether it actually instantiates another pattern B even though there is an instantiation relation from A to B. This is because instantiation is not always automatic and sometimes other patterns are needed for the instantiation to occur. Since the relations do not carry enough information i.e. they do not state the required and necessary conditions for instantiation, modulation and potential conflicting of a pattern, knowledge as to when a pattern is automatically instantiated for example, can only be deduced by reading the descriptions of the related patterns.

Secondly, although patterns form a hierarchy based on their instantiation relations, it does not mean that there is just a single hierarchy in a game, but in fact multiple pattern hierarchies can be found in any given game (Björk and Holopainen, 2005a, 37).

Thirdly, patterns are applied in limited design spaces. Björk and Holopainen make an explicit note of this only in regards to modulation (Björk and Holopainen, 2005a, 35), but it applies to instantiation and potentially conflicting relations as well. Limited design space means that pattern's effects are limited to a concrete aspect of gameplay.

As an example, in the classic roguelike roleplaying game, *NetHack*[8], players spend most of their time exploring the dungeon and when they run into a hostile opponent, they may have to fight as well. In this case, pattern *Combat* (Björk and Holopainen, 2005a, 145-147) is applied only in the conflict resolution forming a limited design space. Of course there can be other patterns also that have meaning only in this limited aspect of gameplay, such as *Achilles' Heels* (Björk and Holopainen, 2005b), or it might be that patterns applied in other parts of the game suddenly have a different meaning in this situation.

3.5.3 Uses of game design patterns

Björk and Holopainen offer four ways in which they see game design patterns as supportive of game design (Björk and Holopainen, 2005a, 45-48):

1. Idea generation. A designer could use patterns by simply choosing a random set of patterns and trying to see if those patterns could form a game. Secondly, a designer could start with an idea i.e. a set of patterns, and start expanding it by adding or deleting patterns, or even modifying the existing ones.

2. Development of game concepts. Probably the most useful and definitely the most central use from the viewpoint of this thesis. Game design should start with a game concept, which is to say that there is a formulated basic idea for the game with varying depth of detail as to how the game should be done. After the concept is decided on, a designer can describe this idea in terms of patterns which results in a structured description of the game, which in turn can be iteratively detailed more accurately by adding, deleting and modifying patterns in it.

3. Problem solving. Although not meant to be purely problem solving tools, patterns do also have their uses in this regard also. They can be especially useful in identifying the causes of problematic aspects of gameplay. Once a cause has been found, patterns sometimes also contain information as to how to fix or at least bypass it.

[8] http://www.nethack.org/

4. Communication. Structured descriptions of game design is very useful in conveying information on the game to other designers, game implementors and people of interest in general.

The focus of this thesis has been in analysing games as entities defined, influenced and produced by rules, so how do game design patterns fit into this viewpoint? As opposed to the 400 Project, game design patterns do not provide clear sets of rules as to how to implement a given aspect of games, but a pattern more or less describes the contexts in which a certain game mechanic can be used and what possible consequences it produces. In other words, game design patterns provide designers with insight into the question: "If I were to implement this game mechanic, how would its rules be intepreted in general and how would it effect the whole of this design?" Looking back to the characteristics of rules and the theory of law discussed in section 2.2.1, legal principles work in the very same manner by giving information to both the enactors and the users of law as to what kinds of laws can be enacted and how a law should be interpreted. Therefore game design patterns could be termed as the *principles of game design*: established but non-binding explanations underlying the formulation of game rules.

As principles, game design patterns are a source of knowledge into the established practices of game design containing descriptions of common ways of producing desired meaning in the game system. As has been argued earlier in this thesis, the successfulness of an explicit operational rule is often dependent on the various implicit rules affecting it. Identifying these implicit rules as early as possible is crucial for the game designer and game design patterns are well suited for this. In addition to describing textually the conditions and consequences of use, each pattern also carries the structural information with it, pointing both to the possible means of modifying these implicit effects, but also to the potentially emerging properties resulting from the implementation of a given game mechanic. So, as to sum it up, *game design patterns aid in controlling the effects of implicit rules and the formation of emergent properties*.

4 Computer-aided game design

4.1 Computer-aided software engineering

Game design is largely a process of iterative refinement, where the concept and mechanics are modified and re-tested over and over again until a satisfactory result has been reached. The basic tool for this kind of refinement is prototyping, which allows the designer to simulate the interaction mechanics of the game and can be built with virtually anything from paper-clips to more sophisticated computer simulations (Fullerton et al., 2004, 157-195). Although absolutely necessary and a powerful tool for game design, there are some restrictions to prototyping that need to be understood. Firstly, prototyping is better suited for testing the mechanics of the game than for instance the social networking between the players. Secondly, prototyping is useful only after an initial design has already been created. Thirdly, depending on the game, prototyping can be a time-consuming effort especially if the game design hasn't been properly fleshed out and the game is even moderately complex.

Of course the restrictions of prototyping can be minimised by creating more flexible testing environments, but it does not change the fact that prototyping is a tool for *testing* design decisions but it is not very suitable for *estimating* their effects before making them. This is the area to which *CAGE* is intended for. It is not a mechanics designing tool or a testing tool, but a tool for understanding and refining the broad view on the design.

The basic purpose of *CAGE* is to allow designers to build a structural drawing (see e.g. Figure 4.14) of the game design in terms of the various meaning producing elements within the game. The drawing is not a process model of the game or a model of the system interaction but a conceptual model of the description of the game i.e. it is a visualisation of the description.

In more than few aspects *CAGE* is quite similar to a group of software tools, known

as CASE-tools[1], which are used in software development for conceptual modelling of software. Although games and software are not the same thing and their design process is quite different, the same principles can be applied for creating a design tool in both cases.

Broadly put, CASE-tools are documentation systems. Software is written and defined using a programming language such as C++ or Python and the basic form of documentation is the source code, where the programmers specify written descriptions for the different parts of the program such as for example classes, methods and modules. This documentation is certainly an indispensable aid for understanding how a specific part of the software works, but for understanding how the whole program works and especially, why the program is designed the way it is, reading the source code documentation can be a truly time-consuming and tedious task.

Understanding a software system is critically important during development, when the programmers need to know what they are doing, and during maintenance operations at the later stages of the program's life cycle (Mayrhauser and Vans, 1995). One way of improving comprehension is visualisation (Price et al., 1992), which is the mechanism used by most CASE-tools. Modern-day software engineering uses a variety of abstraction models, such as object-oriented class hierarchies, architectural styles and software design patterns, to describe and even simulate the system functionality before implementation. Using a suitable notation or specification language such as UML (Unified Modelling Language, see for example (Arlow and Neustadt, 2002)) or some other symbolic modelling technique like ERM (Entity-Relationship Model, see for example (Chen, 1976)), these abstraction models can then be used as a basis for a structural visualisation method such as class diagrams (for example, see 4.1) or ER-diagrams.

A basic requirement for a CASE-tool is to support a chosen design methodology. In addition to visualisation aids, in software engineering this support can include also automated checks for ensuring completeness and consistency of the design (Jankowski, 1997). A desirable feature for a computer-aided design tool is clearly the ability to adapt also to different kinds of design methods.

[1]Computer-Aided Software Engineering

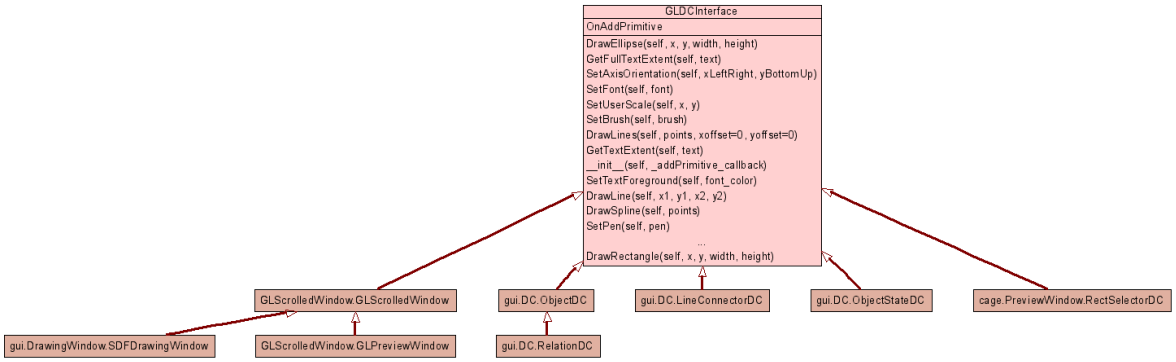

Figure 4.1: Example class diagram using UML-notation.

4.2 Conceptual modelling of game design

The general idea behind *CAGE* is the same as in CASE-tools: support a chosen design method and improve design comprehension by visualising the design using a structural schema - in fact *CAGE* uses a somewhat similar approach to a software tool described by Schauer and Keller in (Schauer and Keller, 1998), which uses software design patterns[2] as the structural schema. However, as opposed to typical software engineering process, where the documentation is mainly aimed for the maintenance phase, the target audience for *CAGE* are the people responsible for a game's development in the first place.

In game development, the typically used ways of communicating the design to other members of the development group are game design documents and prototyping. Although both are essential in a game development process, they are not perfectly suited for easy comprehension. A game design document is a specification of the game defining loads of information simply not necessary for understanding the gameplay and at the same time, it usually does not describe reason for certain decision. Prototyping as a testing and refinement tool, on the other hand, is more useful in describing individual game mechanics than the whole game. Therefore, *CAGE* attempts to improve game design comprehension by providing a description model between these two with the goal of providing also reasons for various design decisions - not focusing purely on *what* the design is but also *why* it is.

[2]See *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma et al. (1995)

The abstraction model used in *CAGE* for describing the design is discussed and defined in chapters 2 and 3. This model is basically divided into two interlaced layers with different levels of abstraction. At the higher level, game design is understood as consisting of a group of meaning-producing elements and their interrelations. As such, these elements can be used to specify a skeleton, or architecture, for the design. The elements can be connected to other elements and they can also be inside other elements. The lower level complements and describes the higher level more thoroughly consisting of more detailed aspects for each element and stricter rules for describing them and their inter-relations. This layer is defined with game design patterns.

Successful visualisation of these two layers is clearly a critical design requirement for *CAGE*. As pointed out by Price et al., a "good [..] visualization system allows the researcher make discoveries not otherwise possible and provides him with a powerful new interface to his data" (Price et al., 1992). This is also attempted in *CAGE*. In game design, it is especially important to see how complex one's design is and what are the consequences for modifying it. In *CAGE*, these two requirements are featured in the lower abstraction level. As a game-element may consist of a group of patterns, adding new patterns into the group may lead to three consequences:

1. There may be potentially conflicting patterns within the group.
2. There may be possibilities of modulation within the group.
3. There may be new, automatically instantiated patterns within the group.

Therefore a new requirement was added:

- Support the visualisation of the consequences of design decisions.

Although very important indeed, this requirement turned out to be quite problematic due to the nature of the patterns. Instead of being a formal language with simple grammar rules, game design patterns are a quite large collection of semantic heavy entities with rather complex rules of production. Since there are no rules to state when a pattern is absolutely instantiated by a pattern or patterns, then all instantiation rules need to be handled in similar fashion. Therefore there are two types of patterns in *CAGE*:

actualised and *potential*. When a pattern is added to or actualised in the design, all the patterns it instantiates are also added to the design, but only as potential patterns. This way it is left to the designer to decide which patterns may actually be instantiated.

As an example, Figure 4.2 shows the effects of inserting a single pattern, *Aim&Shoot*, into a design. The pattern is added into a container representing some mechanic in the design. The added pattern is shown as a green ellipse and the patterns it potentially instantiates are shown as white ellipses. Instantiation-relations are drawn in blue and modulation in green arrows.

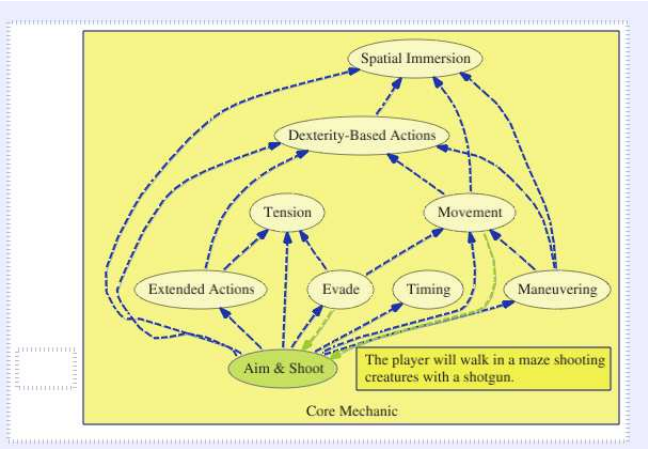

Figure 4.2: *Aim&Shoot*-pattern inside a container.

Figure 4.3 shows a good example of the complexities of using game design patterns as a description language. Here a new pattern, *Movement*, is added to the same container. As an effect this creates six new potential patterns and adds 21 new relations to the drawing. In order to be comprehensible at all, even an actualisation of two patterns can place heavy requirements for the layout-algorithm, let alone the actualisation of five or more patterns. This presented problems that cannot be said to have been solved satisfactorily.

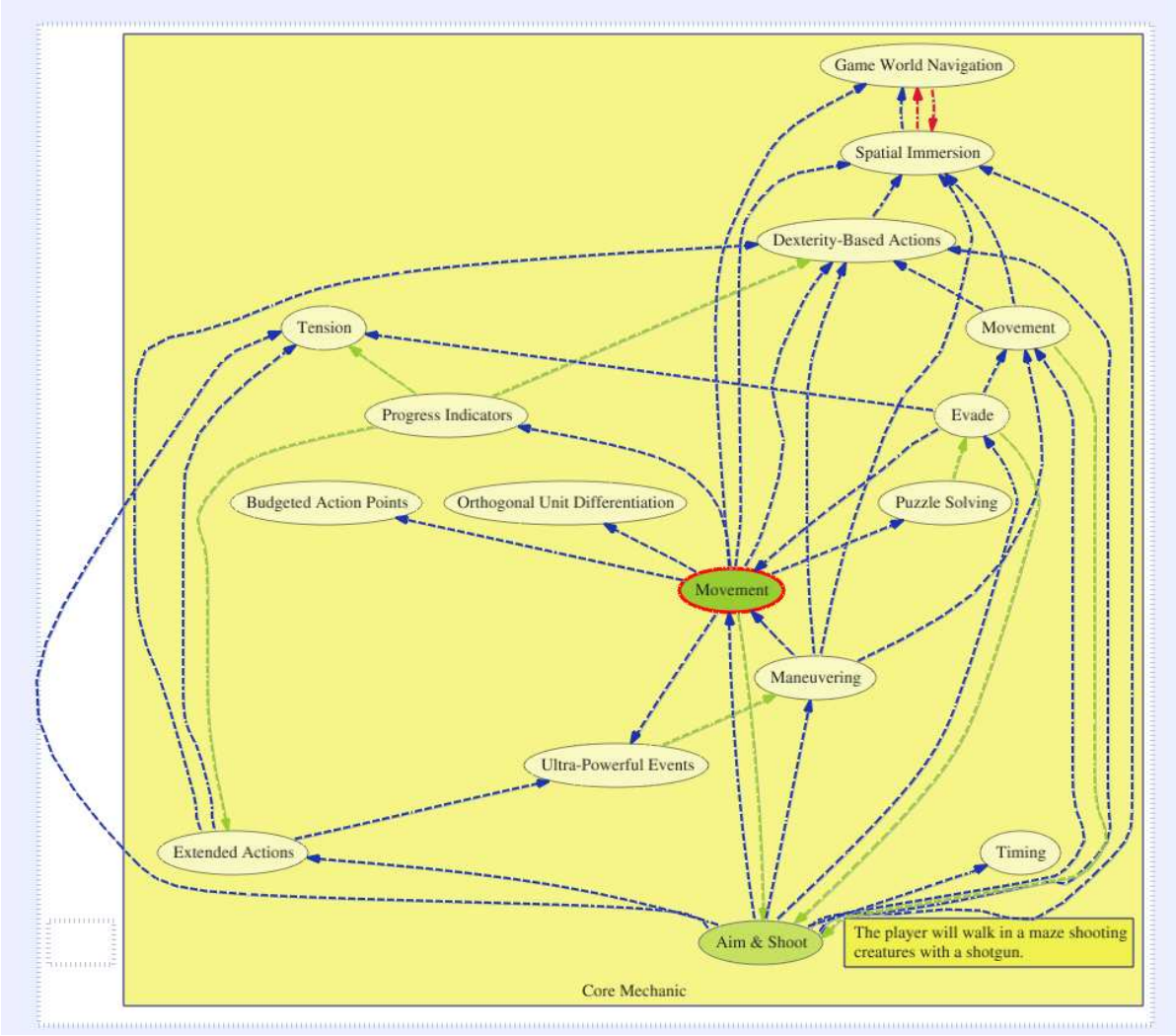

Figure 4.3: The same container as in Figure 4.2 but with *Movement*-pattern inserted into it.

CAGE requires a layout-algorithm capable of visualising very complex directed graphs in a manner that would still be useful for the designer. The best non-proprietary solution that could be found was the *dot*-algorithm in Graphviz, which fills the requirements otherwise except that it does not support incremental changes, but calculates the layout from scratch every time a slightest change is introduced to the design.

Method-support in *CAGE* consists of two basic requirements:

- Support iterative design.

- Support multiple structural schemas.

CAGE follows a top-down -approach to game design where the general structure of the game is laid out first and incrementally defined in more detail, thus supporting iterative approach to design. Structural schemas at the higher level of abstraction can be freely modified and changed by the user. However, the lower abstraction level of game design patterns cannot be totally changed although some game design methods would require it. Although the patterns collection can be freely modified and even the current relationship-model in game design patterns can be replaced, the patterns still cannot be changed to, for instance, the 400 rules approach. Neither is the addition of new layers easily supported as would be required for instance in creating more abstracted collections of patterns in light of the MDA-approach.

4.3 Structure of *CAGE*

4.3.1 Higher level structure

CAGE is a tool to explore the possibilities of using game design patterns as the specification language for game designs. The program is written in Python[3] using *wxPython*[4] for the graphical interface.

The software consist of three separate components, *cage*, *sdf* and *opengl*, which apart from the two embedded external libraries, *mfgraph*[5] and *ftgl*[6], were all written by the author during the process. As shown in Figure 4.4, the opengl-package is embedded in the sdf-package, but it could also be used separately for drawing 2D-shapes in wxPython-application.

[3]http://www.python.org/
[4]http://www.wxpython.org/
[5]http://mfgraph.sourceforge.net/
[6]http://homepages.paradise.net.nz/henryj/code/#FTGL

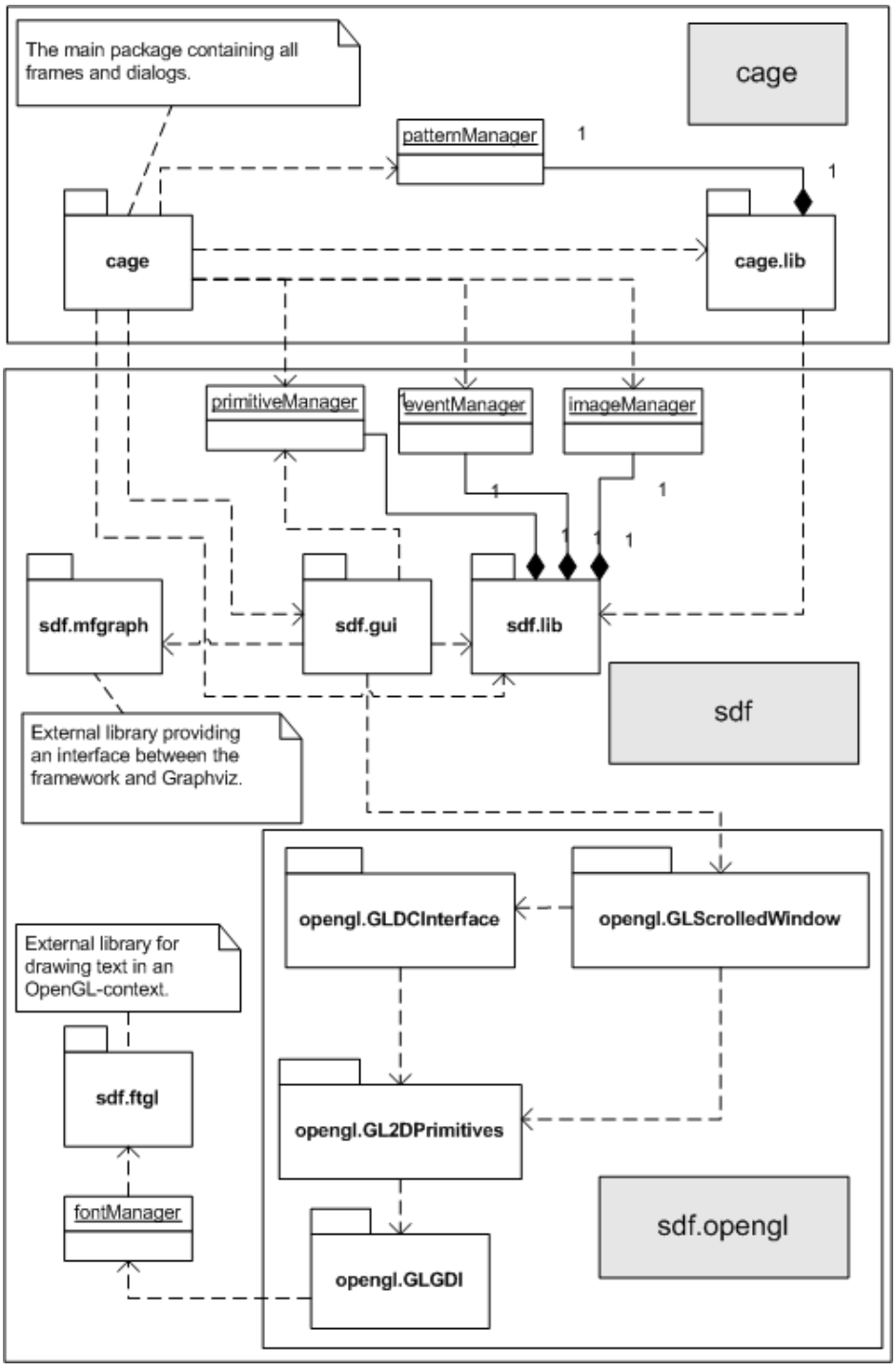

Figure 4.4: The high-level structure of *CAGE* as a UML-diagram.

In terms of functionality, the *sdf*-package provides the general business logic and data persistence, and *cage* the user interaction and specialised services such as game design pattern -management, which should not be part of a general framework. *Opengl* provides the lower-level drawing functionality. Software-wide access to shared resources such as events, data primitives, images, pattern-collections and opengl-compatible fonts is provided via singleton managers.

4.3.2 *sdf*-package

The *sdf*-package, which stands for *Structured Drawing Framework* provides a general framework used for visualising structural data. The reason for implementing the drawing functionality as a framework was not only that it would produce more reusable code, but for two key issues:

- Due to the experimental nature of *CAGE*, the program should be as flexible as possible.
- The set of meaning-producing elements defined in this work is not meant as the only possible set, but should be modifiable by individual designers according to their own preferences.

As a result, the framework allows for easy changes within the drawable element set by keeping all the structural and visual restrictions in a single xml-file, where it is easy to change the outlook of the elements and the rules for drawing them. One can remove the current element and relation types, create new ones and modify existing ones. Only restriction to this is that the current version of *CAGE* treats game design patterns as a special type which has to be defined within the structural definitions. *SDF* as an individual framework is not subject to this restriction.

There are three basic drawable element-types:

- *Container*-element, which can hold other containers and objects.
- *Object*-element, which is a single entity.

- *Relations* for connecting elements. Relations are directional i.e. they are from one element to another element.

All element-types have some additional properties such as *shape*, *text-colour*, *font-size*, *colour* and so on forth. All these properties can be individually defined in different states, such as *selected*, *not selected*, *potential* and *actualised*. Containers can define the kinds of elements they are allowed to contain and relations can define the elements they can connect to. Objects and containers also have a *name* and a *description* with revision history.

The basic element-types are used to construct a schema for the structure the program wants to represent. One can create a number of different containers, objects and relations with rules for their usage. As an example, in the case of *CAGE*, its current schema is made of the following elements:

- *Resources, Social Interaction, Conflict, Actions, Goals, Representation* and *Information* for game elements. These are containers.
- *Patterns* as the properties or aspects of the elements. These are objects.
- Relations *Connects*, *Instantiates*, *Instantiated By*, *Modulates*, *Modulated By* and *Potentially Conflicts With*. *Connects* is just an ordinary connection between two elements and the rest are more specifically defined in section 3.5.

Package *sdf* is the heart of *CAGE* and provides quite a lot of functionality. There are classes for data-handling, project management, uniform event handling within the framework, GUI-elements with drag-and-drop support for the basic elements, a specialised styled text editor with its own markup-language and so forth. The framework consists of five subpackages *sdf.lib*, *sdf.opengl*, *sdf.gui*, *sdf.mfgraph* and *sdf.ftgl*, of which the last two are external libraries embedded into the framework for ease of use.

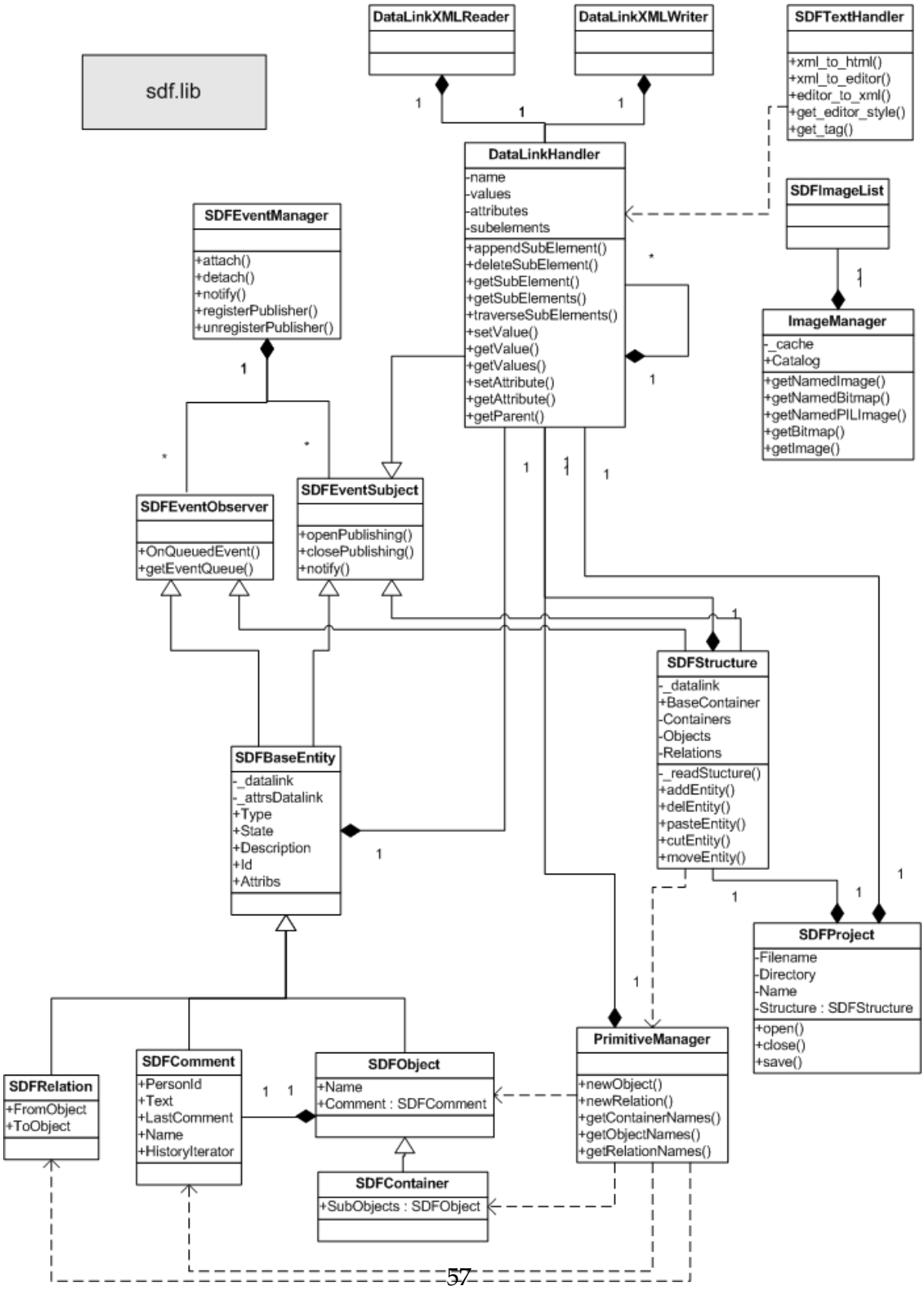

Figure 4.5: UML-diagram of *sdf.lib*.

As shown in the Figure 4.5, at the core of *sdf.lib* is the *DataLinkHandler*, which is part of almost every class within the package. This class defines the interface for handling and storing data in xml-files using a similar approach to XML DOM, but with an easier use and *sdf*-specific event-handling capabilities. A single *DataLinkHandler* is normally used to provide business objects with an individual handle to a larger dataspace such as a drawing structure saved to an xml-file. Objects can then change their own data without having to know anything about other objects except their own sub-objects.

Another central element of the framework are the event-handling classes using a *Subject-Observer* software design pattern -style (Gamma et al., 1995, 293-305) approach with a centralised, singleton *eventManager*, which is an instance of *EventManager* for publishing and relaying events from subjects to observers. Any class that subclasses *SDFEventSubject* can register itself to the *eventManager* and start publishing its events, which in turn can be observed by any descendant of *SDFEventObserver.*

Sdf.lib also defines the business logic for handling the aforementioned schema for the structural elements in form of *SDFBaseEntity* and its descendants. Access to these entities is provided via a singleton *primitiveManager*, which is an instance of *PrimitiveManager*.

The drawings i.e. the conceptual models are stored and can be accessed through an *SDFStructure*-object, which is provided by an *SDFProject*-object handling other project-specific functionality such as creating, opening and saving projects. Finally, *sdf.lib* defines classes for managing image-files, i.e. *ImageManager* and *SDFImageList*, and an *SDFTextHandler*-class for handling the styled text markup -language in the framework.

The *sdf.gui* -subpackage provides the *SDFDrawingWindow*, which is the cornerstone of the workspace in *CAGE.* This class can read project structures, call *Graphviz* through mfgraph and draw the resulting layout in accordance with the given structural element schema. User can then manipulate this structure in various ways such as for example drag new elements into the structure or move elements from one container to another.

The framework relies on two external packages and programs for some functionality. Of core importance for the whole framework is the *Graphviz*[7] - graph visualisation tool, which handles the layout-calculations for all structures drawn with *sdf*. The *Graphviz* is currently interfaced using *mfgraph*.

[7]http://www.graphviz.org/

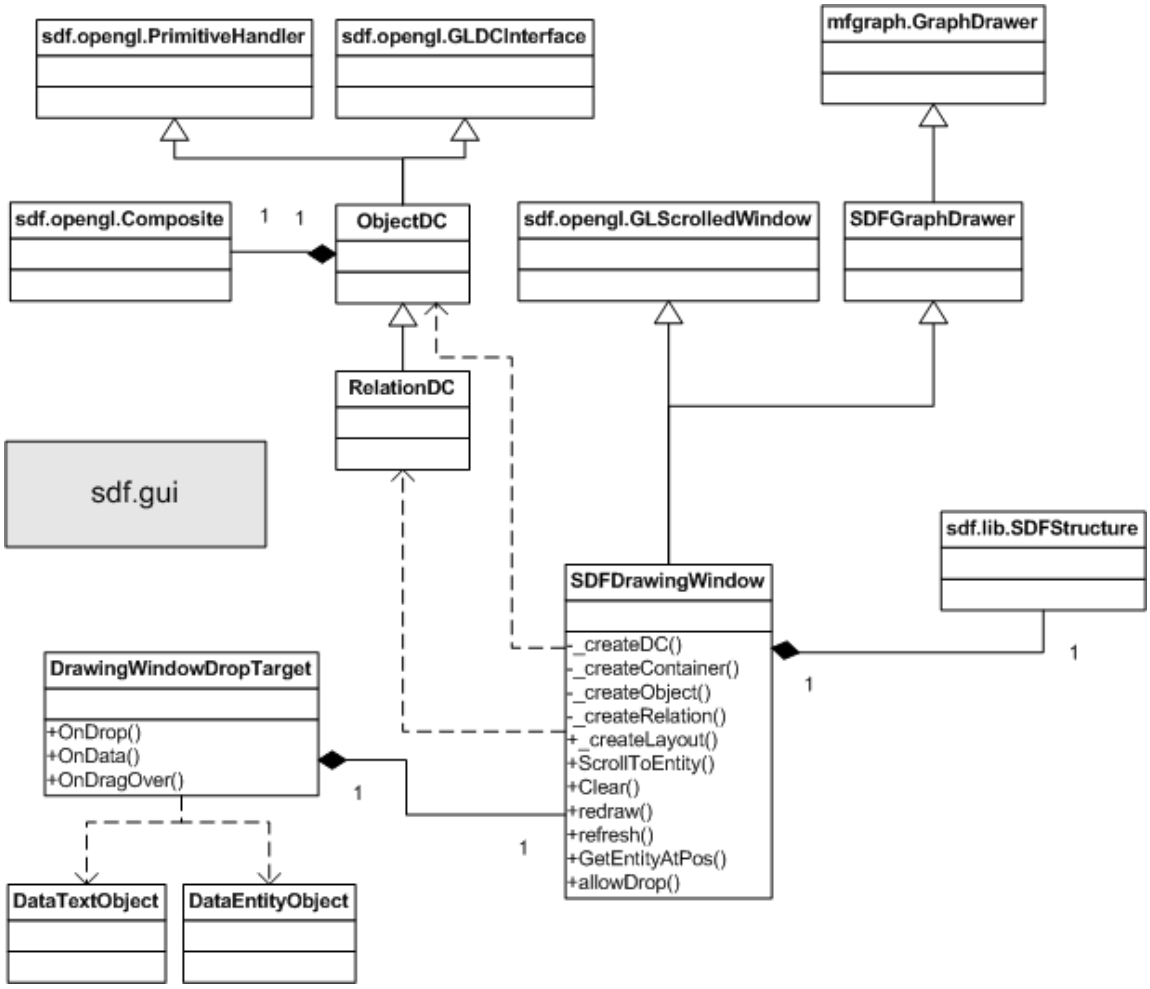

Figure 4.6: UML-diagram of *sdf.gui*.

4.3.3 *opengl*-package

The *opengl*-package (see Figure 4.7) for drawing 2D-graphics with *OpenGL*. The package provides two user interface elements:

1. *GLScrolledWindow*, which implements an almost full *wx.DC*-interface via *GLDCInterface* for drawing operations and offers also the same functionality as *wx.ScrolledWindow*. All drawables, e.g. *Circle*, *Line*, etc., are object primitives which can be grouped and easily moved.
2. *GLPreviewWindow*, which is inherited from the *GLScrolledWindow*. As the name suggests, this class is intended for previewing the drawings, but also offers the capability for exporting the drawings in other formats such as PNG (Portable Network Graphics) and EPS (Encapsulated PostScript).

Since the *GLScrolledWindow* uses the wx.DC -interface, the whole opengl-package should be quite easy to replace with another library such as the standard *wxPython* device contexts or another more specialised drawing library. For the same reason *opengl*-package uses its own event-handling instead of the general event-handling mechanisms provided by and used in the *sdf*.

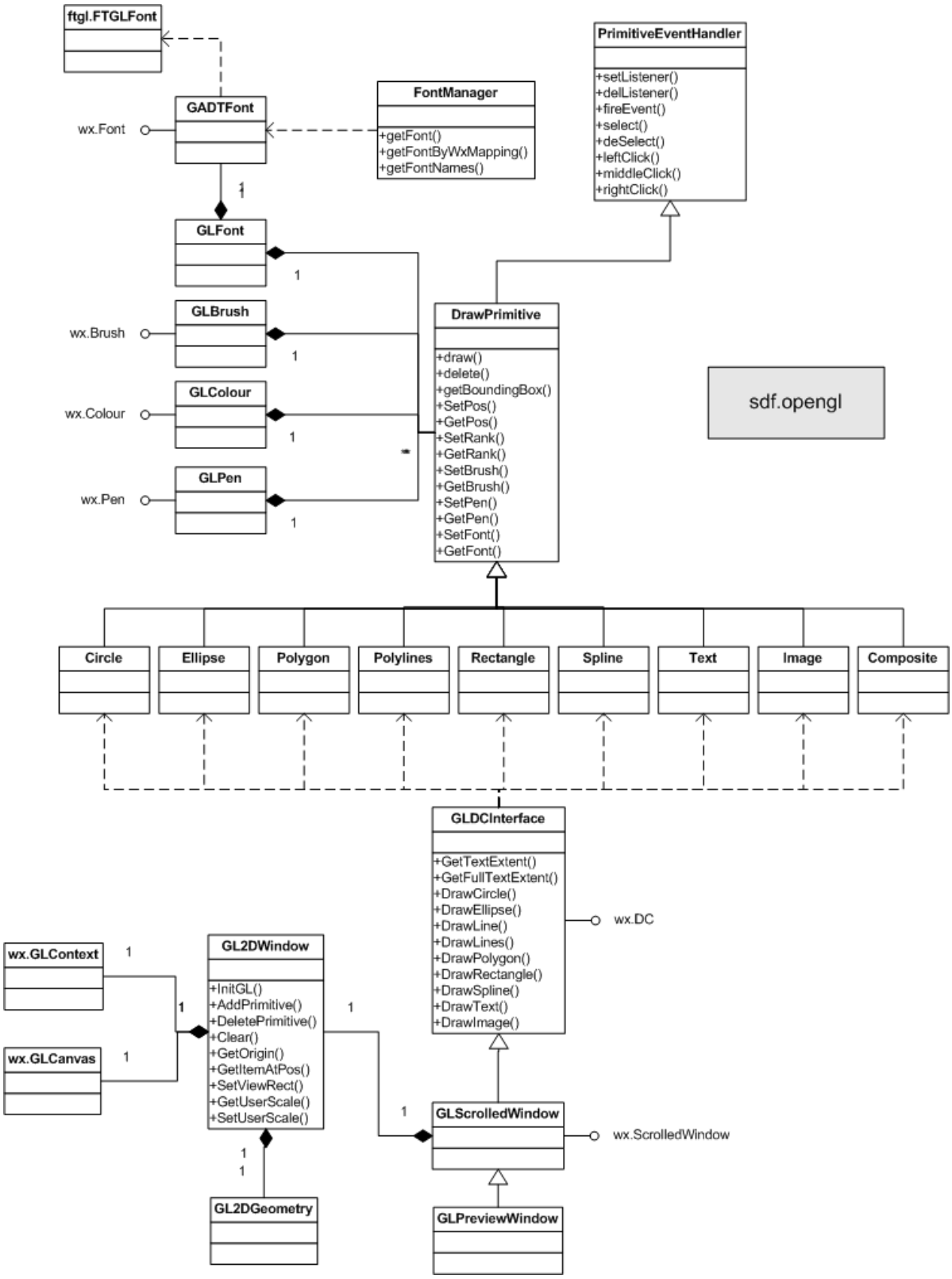

Figure 4.7: UML-diagram of *sdf.opengl*.

OpenGL was chosen for the actual drawing since it was known that the graphs could get quite large and an efficient graphics library was required especially for zooming and panning of the drawing. The 2D-graphics capabilities of *wxPython* were certainly not enough. Another important reason for choosing *OpenGL* was its good support for transparent colours, which was lacking at the time in *wxPython*. Transparency is used in *CAGE* to visualise depth for elements in containers by blending the overlapping colours so that the elements inside a container seem darker than their parent. See figure 4.8 for an example.

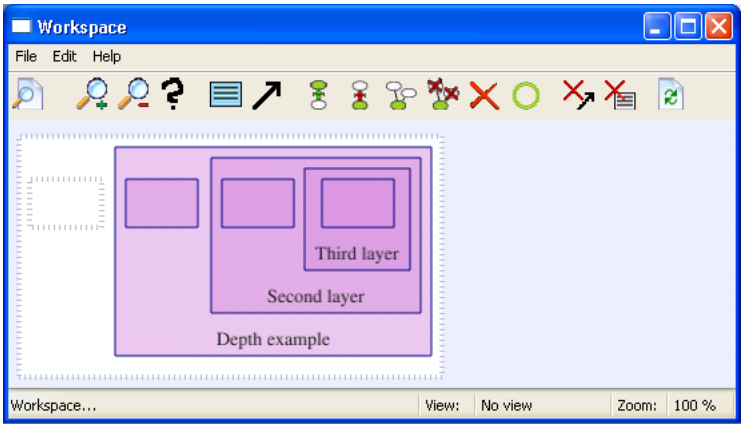

Figure 4.8: Using colour transparency for depicting element depth inside a container.

Although *opengl* can be used independently, in *CAGE* it is integrated into the *Structured Drawing Framework*. The opengl-package currently uses one external package: the *ftgl* -*OpengGL* font rendering library for drawing text into drawing contexts.

4.3.4 *cage*-package

The graphical user interface and the interaction logic is defined in the *gadget*-package, which uses the *sdf* heavily. The interface (Figure 4.9) is an SDI (Single Document Interface) -type having four major components:

- The main window for controlling design projects.
- The resource window providing access to game design patterns.

- The structure view with a hierarchical description of the structure in the currently open design.
- The workspace for the actual designing.

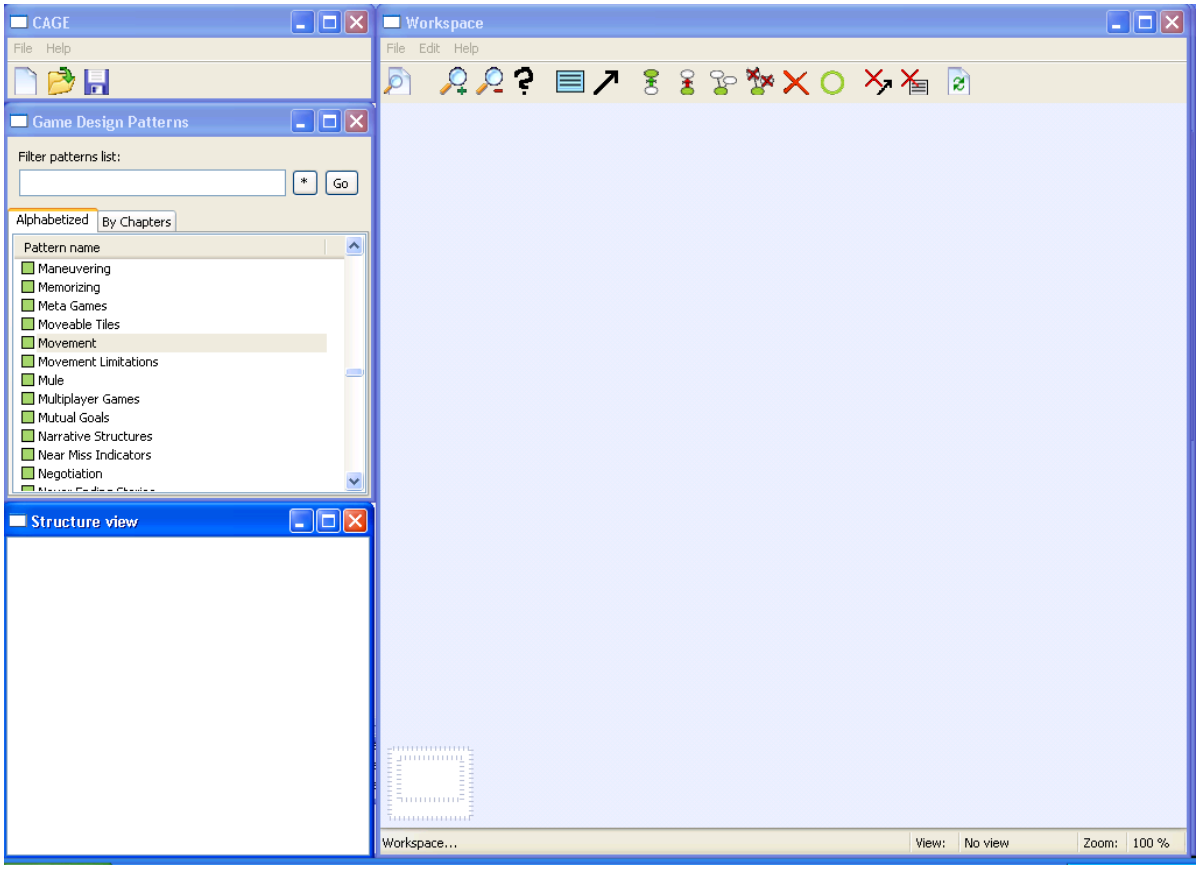

Figure 4.9: *CAGE*'s main view.

In addition to these, there is also an editor for controlling collections of Game Design Patterns. The base collection is the one supplied with the book Patterns in Game Design, but the editor allows also the creation of new collections and merging of old ones. One can create new patterns, modify and delete old ones. There are also crude mechanisms for version control.

The SDI-type interface was chosen primarily because it was clear that the structural drawings could be huge and users might benefit from the possibility of using multiple

displays. With hindsight, this was not an altogether good decision as the user interface is now cluttered and harder to control. However, if needed this can be changed to an MDI-type interface with relatively small work.

4.4 Case study: *Unintelligent Designer*

In order to test out the usefulness and applicability of *CAGE*, an exsting game idea was developed further. The idea was a rejected concept created within the Intergrated Project on Pervasive Games, IPerG[8]. *Unintelligent Designer* is a simple pervasive game[9], where players compete over limited but shared resources.

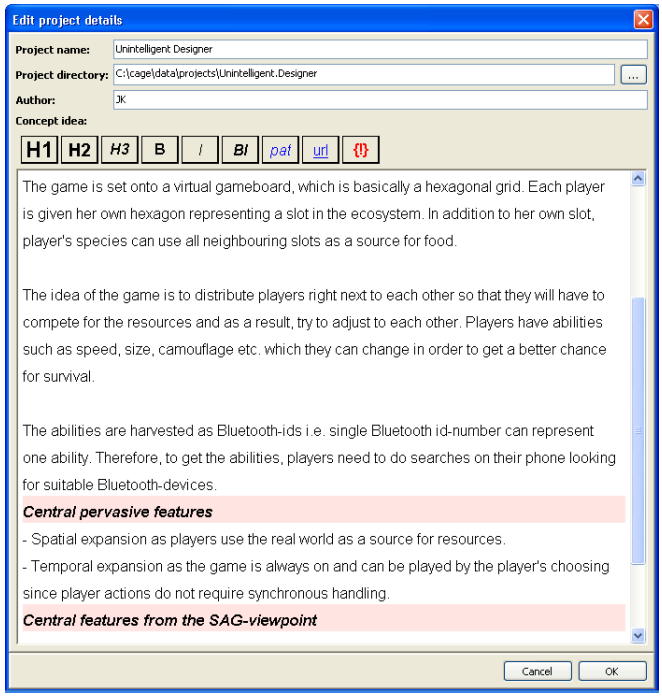

Figure 4.10: Project creation dialog with details concerning the game filled in.

First a new project was created and its details filled out (see Figure 4.10). The game idea is as follows:

[8]http://www.iperg.org/

[9]Pervasive games expand the gaming experience in to the physical world. See IPerG-deliverable D5.3b: Domain of Pervasive Gaming (Montola et al., 2006) for a profound definition of pervasive games.

Players form a system of herbivores and carnivores all trying to survive and spread further throughout the world. Each player starts with a single species which is either a herbivore or a carnivore up to the player's own choosing.

The game is set onto a virtual gameboard, which is basically a hexagonal grid. Each player is given her own hexagon representing a slot in the ecosystem. In addition to her own slot, player's species can use all neighbouring slots as a source for food.

The idea of the game is to distribute players right next to each other so that they will have to compete for the resources and as a result, try to adjust to each other. Players have abilities such as speed, size, camouflage etc. which they can change in order to get a better chance for survival.

The abilities are harvested as Bluetooth-ids i.e. single Bluetooth id-number can represent one ability. Therefore, to get the abilities, players need to do searches on their phone looking for suitable Bluetooth-devices.

The design was started by going through the idea description and defining game's the most central elements in terms of the used structural schema. These were then added into the structure (see Figure 4.11). As a primary goal, players all try to survive and the only way to affect this is by collecting new ability points and thereby adapt one's species to its environment. Each player has an individual ecological slot, but this same slot is also shared as a source of food by other neighbouring players.

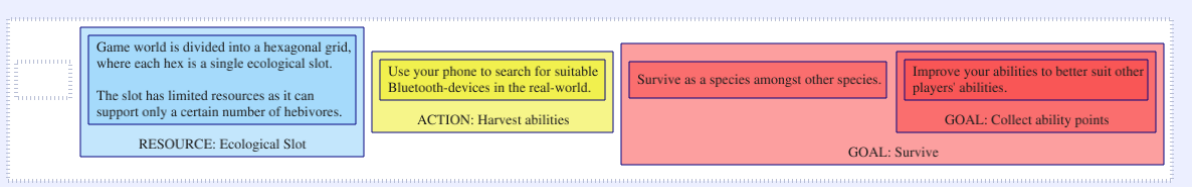

Figure 4.11: The central elements laid out in the structure.

With the core elements in place, it was time to start adding aspects for each element. Each aspect is defined by a game design pattern. Figure 4.12 shows the situation after adding altogether eight aspects to the three elements. Those patterns that are shown as

green ellipses stand for actualised patterns i.e. aspects that were deemed to absolutely belong to the element. Some of the potential patterns created automatically during pattern additions were deleted right away as it was clear that they were not part of the element. A group of patterns also have comments, the white rectangles connected to a pattern, attached to them describing the aspect in more detail.

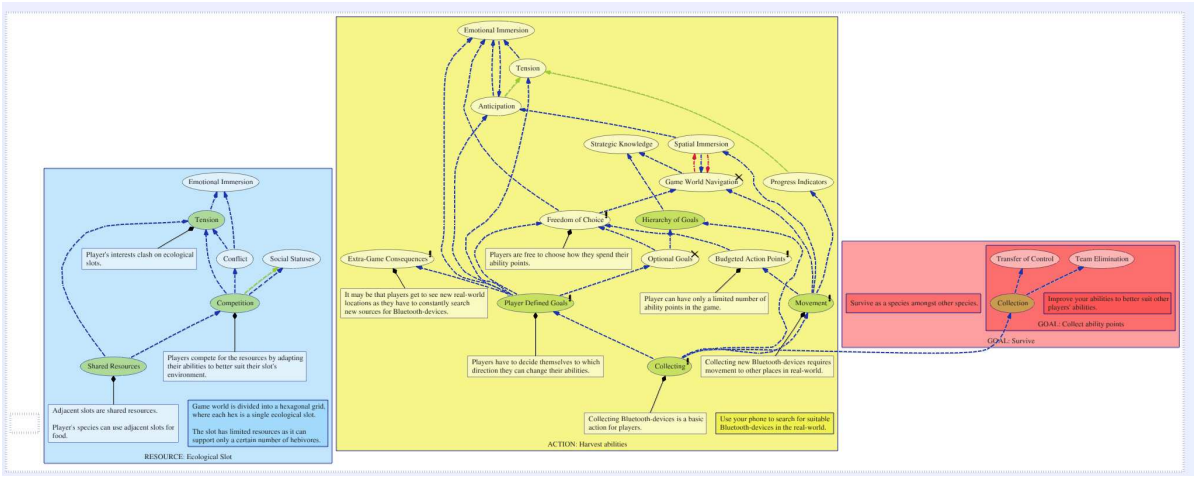

Figure 4.12: Initial refinement of the elements.

The Resource-element for limited food clearly had an aspect, *Competition*, that should not be there. Of course competition is linked to the scarcity of the resource, but it is not an aspect of the resource itself. Therefore, a new element type, *Conflict*, was added to the structure to describe the conflict between players over the limited food supply. *Competition*, *Tension* and the potential aspect, *Conflict*, were dragged into the new element. The resulting structure is shown in Figure 4.13.

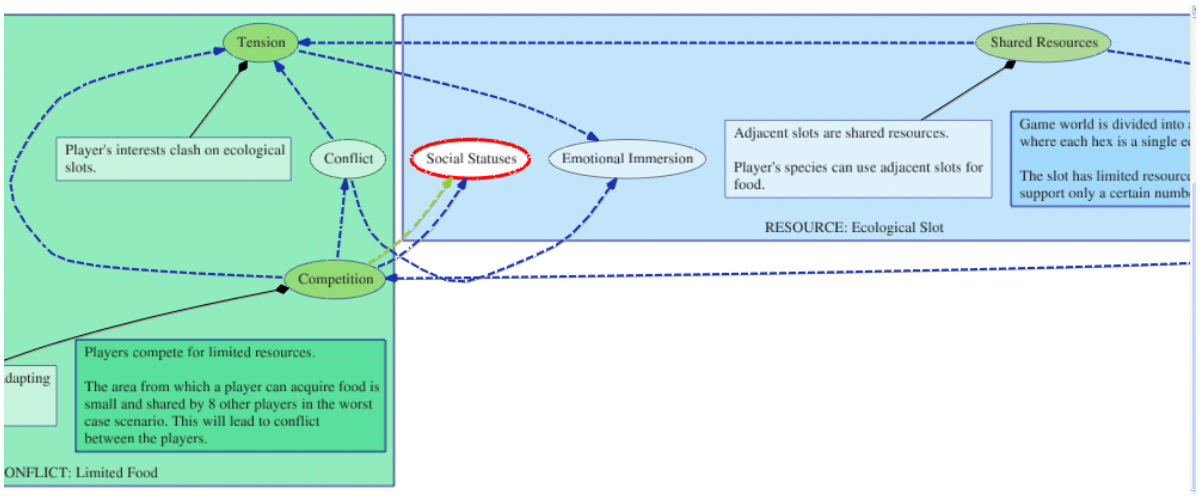

Figure 4.13: *Conflict* added to the structure.

After some reorganisation, the initial core structure was beginning to be ready. Naturally this didn't describe the game completely, but only partially. Many of the elements could have been overloaded with patterns, but then it would be much harder to see the core structures in the game. In game design and especially in the conceptualisation phase, the goal is not to describe the game in a total fashion, but to comprehend the game concept more clearly. Out of that comprehension rises the ability to make good decisions on improving the design. Figure 4.14 shows the core elements and some connections between them.

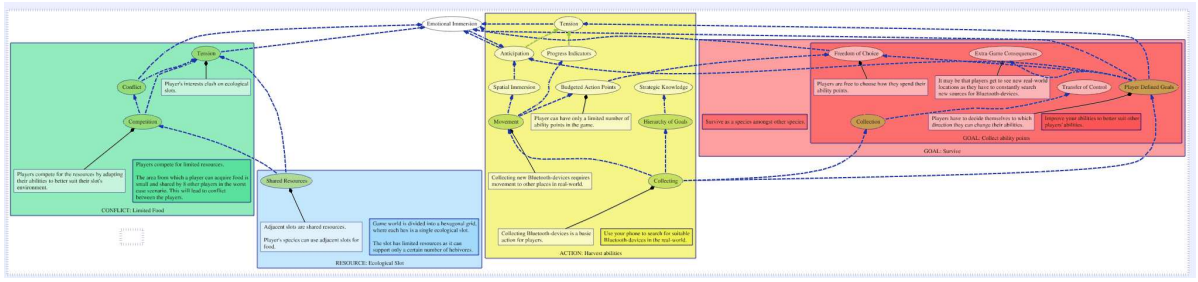

Figure 4.14: The initial structure is ready.

The description of *Unintelligent Designer* is thus purposefully made in a more broader and general way than might be required in a game analysis. For initial concept design, it offers a good starting point. First of all, it gives a documented view on different key

elements within the game, making it easier to communicate the design to other people. Secondly, it forces the designer to view the concept in a more explicit and analytical manner thus deepening the developer's understanding of the concept.

In terms of causes and consequences for the design choices it gives a good initial view, which can later be refined and further developed if needed. For instance, one can see from the design that *Emotional Immersion* may be instantiated by a number of elements and that the *Tension* stemming from the *Player Defined Goals* can be modulated by the *Anticipation* and *Progress Indicators* linked to the action of harvesting the Bluetooth IDs. A structural drawing such as this allows a designer to identify how different elements are connected to each other and how one can modulate the experiences created by these elements. Should the designer now wish to heighten the players' *Tension* for reaching their goals, she could use the *Anticipation* and *Progress Indicators* as tools for it.

5 Conclusions and further study

As the tool it was meant to be, *CAGE* is by no means a total success, although it was always clear that nor would it be. The idea of a computer aided game design is simply too novel an idea and the problem too great for something to be solved in a single master's thesis. However, it could be said that a fair amount of issues related to the topic were uncovered and at least partly, the initial research question was answered.

Game design patterns can be used for visualised game design, but with the approach taken in this work, they are not perfectly suited for it. They can be really helpful in terms of guiding the designer to new inventions but at the same time, they can be unhelpful especially in the hands of someone who does not have good knowledge on game design patterns. The biggest problem with them is the inherent complexity of the basic collection resulting in almost uncontrollably complex structures within the model used in *CAGE*. Although this can be compensated to a certain extent with user interface improvements and user's own expertise, whether it can be totally overcome, can not be satisfactorily answered with this master's thesis.

The experiences from using the program suggests that it may be questionable whether the high level of granularity and specificity of the patterns is really necessary or fruitful in game design. In fact, it could be more fruitful to raise the level of abstraction and restrict the size of the patterns collection. One way to do this could be using the dynamics in MDA described in section 3.4.2. Defining the dynamics as collections of patterns would introduce a new abstraction layer between the current element-pattern -distinction and probably make the program more usable especially in game design, but also in analysis, as it would allow the user to work in a higher abstraction level and switch to a more detailed level when needed.

As a static model, the chosen abstraction model on the whole was not perfectly suitable for game design. In order for a game design tool to be very useful, it should probably take a more dynamic approach allowing the modelling of player interaction in some kind of a temporal context. However, as pointed out already, as a game analysis and

refinement tool *CAGE* can be useful.

The user interface and the visualisation mechanism will need some improving. Looking at the list of desirable properties in a visualisation tool by Young and Munro in (Young and Munro, 1998), it is clear that most of them are poorly supported in *CAGE*. The complexity of the visualisation and the lack of an incremental layout algorithm are definitely crucial problems, but they simply could not be solved with the resources allocated.

To conclude, it might be rewarding to further develop *CAGE* by fixing the user interface. Currently the initial research question of how to create a software tool for game design cannot be fully answered due to the issues in usability. It may be that game design patterns as such are not the ideal means for describing elements in game design, but making the user interface more usable could be beneficial in two ways: First of all, it would allow for better evaluation of the suitability of game design patterns for the task. Secondly, it would probably unveil more problems, but also possibilities in computer-aided game design and analysis. This work has only been able scratch the surface of the issue, but even so the case study and the initial use of the program show possibilities and promise for a game design and analysis software tool.

6 Bibliography

The Collaborative Internation Dictionary of English. URL `http://www.dict.org/bin/Dict?Form=Dict2\&Database=gcide\&Query=interaction`. Accessed on 18.6.2007.

Collins English Dictionary - Third Edition. HarperCollins Publishers, Glasgow, 1991.

FIDE Laws of Chess, 2004. URL `http://www.fide.com/official/handbook.asp?level=EE101`. Accessed on 18.6.2007.

Christopher Alexander. *A Pattern Language Towns, Buildings, Construction*. Oxford University Press, New York, 1979.

Jim Arlow and Ila Neustadt. *UML and the Unified Process Practical Object-oriented Analysis and Design*. Addison-Wesley, Boston, 2002. ISBN 0-201-77060-1.

Chris Bateman and Richard Boon. *21st Century Game Design (Game Development Series)*. Charles River Media, 1 edition edition, Aug 2005.

Staffan Björk and Jussi Holopainen. *Patterns in Game Design*. Charles River Media, Hingham (Mass.), 2005a.

Staffan Björk and Jussi Holopainen. *Patterns in Game Design - companion CD-ROM*. Charles River Media, Hingham (Mass.), 2005b.

Staffan Björk, Sus Lundgren, and Jussi Holopainen. Game design patterns. In Marinka Copier and Joost Raessens, editors, *Level Up Conference Proceedings*, pages 180–193, Utrecht, November 2003. University of Utrecht. URL `http://www.digra.org/dl/db/05163.15303.pdf`. Accessed on 18.6.2007.

Staffan Björk, Jussi Holopainen, and Jussi Kuittinen. Teaching gameplay design patterns. In Igor Mayer and Hanneke Mastik, editors, *Organizing and Learning through Gaming and Simulation, Proceedings of Isaga 2007*, Nijmegen, Netherlands, 2007. Eburon. forthcoming spring 2008.

Peter Pin-Shan Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/320434.320440.

Doug Church. Formal abstract design tools. In Katie Salen and Eric Zimmerman, editors, *The Game Design Reader a Rules of Play Anthology*, pages 366–381. MIT Press, Cambridge (Mass.), 1999. ISBN 0-262-19536-4.

Greg Costikyan. I have no words & I must design. In Katie Salen and Eric Zimmerman, editors, *The Game Design Reader a Rules of Play Anthology*, pages 192–211. MIT Press, Cambridge (Mass.), 1994.

Robert W. Crandall and Gregory J. Sidak. Video games - serious business for america's economy, 2006. URL `http://www.theesa.com/files/VideoGames-Final.pdf`. Accessed on 18.6.2007.

Chris Crawford. *Chris Crawford on Game Design*. New Riders, Indianapolis (Ind.), 2003.

Ronald Dworkin. *Taking Rights Seriously*. Gerald Duckworth & Co. Ltd., London, 1996.

Laura Ermi and Frans Mäyrä. Player-centred game design: Experiences in using scenario study to inform mobile game design. *Game Studies*, 5(1), oct 2005. URL `http://www.gamestudies.org/0501/ermi_mayra/`. Accessed on 18.6.2007.

Noah Falstein and Noah Barwood. The 400 project, 2003. URL `http://www.theinspiracy.com/400_project.htm`. Accessed on 18.6.2007.

Luciano Floridi. Is information meaningful data? *Philosophy and Phenomenological Research*, 70(2):351–370, 2005. URL `http://www.wolfson.ox.ac.uk/~floridi/pdf/iimd.pdf`. Accessed on 19.1.2008.

Tracy Fullerton, Christopher Swain, and Steven Hoffman. *Game Design Workshop Designing, Prototyping, and Playtesting Games*. CMP Books, San Francisco (Calif.), 2004.

Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns Elements of Reusable Object-oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, Reading (Mass.), 1995.

H.L.A. Hart. *The Concept of Law*. Oxford University Press, Oxford, 1997. ISBN 0-19-876123-6.

Johan Huizinga. *Homo Ludens: A Study of the Play Element in Culture*. Routledge & Kegan, Paul, London, 1980.

Robin Hunicke, Marc LeBlanc, and Robert Zubek. Mda: A formal approach to game design and game research, 2004. URL `http://www.cs.northwestern.edu/~hunicke/pubs/MDA.pdf`. Accessed on 18.6.2007.

David Jankowski. How can case help?: a look at the feasibility of supporting structured analysis with case. *SIGMIS Database*, 28(4):33–47, 1997. doi: http://doi.acm.org/10.1145/277339.277343. Accessed on 18.6.2007.

Aki Järvinen. Making and breaking games: A typology of rules. In Copier Marinka and Raessens Joost, editors, *Level Up Conference Proceedings*, pages 68–79, Utrecht, November 2003. University of Utrecht. URL `http://www.digra.org/dl/display_html?chid=05163.56503`. Accessed on 18.6.2007.

Henry Jenkins. Game design as narrative architecture. In Katie Salen and Eric Zimmerman, editors, *The Game Design Reader a Rules of Play Anthology*, pages 670–689. MIT Press, Cambridge (Mass.), 2004.

Jesper Juul. *Half-real Video Games Between Real Rules and Fictional Worlds*. MIT Press, Cambridge (Mass.), 2005.

Bernd Kreimeier. Game design methods: A 2003 survey, 2003a. URL `http://www.gamasutra.com/features/20030303/kreimeier_03.shtml`. Accessed on 18.6.2007.

Bernd Kreimeier. A case for game design patterns, 2003b. URL `http://www.gamasutra.com/features/20020313/kreimeier_03.htm`. Accessed on 18.6.2007.

Marc LeBlanc. Tools for creating dramatic game dynamics. In Katie Salen and Eric Zimmerman, editors, *The Game Design Reader a Rules of Play Anthology*, pages 438–359. MIT Press, Cambridge (Mass.), 2005.

A. von Mayrhauser and A.M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.

Markus Montola, Annika Waern, and Eva Nieuwdorp. Deliverable d5.3b: Domain of pervasive gaming, 2006. URL http://iperg.sics.se/Deliverables/D5.3b-Domain%20of%20Pervasive%20Gaming.pdf. Accessed on 18.6.2007.

David Parlett. Rules OK. A paper presented at the Board Games Studies Conference, Oxford, 2005. URL http://www.davidparlett.co.uk/varia/rulesok.html. Accessed on 18.6.2007.

B.A. Price, I.S. Small, and R.M. Baecker. A taxonomy of software visualization. In *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, volume ii, pages 597–606, 1992.

Andrew Rollings and Ernest Adams. *Andrew Rollings and Ernest Adams on Game Design*. NRG. New Riders, Indianapolis (Ind.), 2003.

Richard Rouse. *Game Design Theory & Practice*. Wordware, Plano (Tex.), 2001.

Katie Salen and Eric Zimmerman. This is not a game: Play in cultural environments. In Copier Marinka and Raessens Joost, editors, *Level Up Conference Proceedings*, pages 14–28, Utrecht, November 2003. University of Utrecht. URL http://www.digra.org/dl/display_html?chid=05163.47569. Accessed on 18.6.2007.

Katie Salen and Eric Zimmerman. *Rules of Play: Game Gesign Fundamentals*. The MIT Press, 2004.

R. Schauer and R.K. Keller. *Pattern visualization for software comprehension*. 1998. URL http://ieeexplore.ieee.org/iel5/5650/15136/00693273.pdf. Accessed on 18.6.2007.

C.E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948. URL http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf. Accessed on 19.1.2008.

Stephen Sniderman. Unwritten rules. In Katie Salen and Eric Zimmerman, editors, *The Game Design Reader a Rules of Play Anthology*, pages 476–503. MIT Press, Cambridge (Mass.), 1999.

Gordon Walton. Bringing engineering discipline to game development, 1998. URL http://www.gamasutra.com/features/19981218/walton_01.htm. Accessed on 19.1.2008.

P. Young and M. Munro. Visualising software in virtual reality. In *Proceedings of the IEEE 6th International Workshop on Program Comprehension*, pages 19–26, Ischia, Italy, June 1998. URL http://citeseer.ist.psu.edu/young98visualising.html. Accessed on 18.6.2007.

José Zagal, Miguel Nussbaum, and Ricardo Rosas. A model to support the design of multiplayer games. *Presence: Teleoperator and Virtual Environments*, 9(5), 2000. URL http://citeseer.ist.psu.edu/zagal00model.html. Accessed on 18.6.2007.

José Zagal, Michael Mateas, Clara Fernández-Vara, Brian Hochhalter, and Nolan Lichti. Towards an ontological language for game analysis. In de Castell Suzanne and Jenson Jennifer, editors, *Changing Views: Worlds in Play*, page 13, Vancouver, 2005. University of Vancouver. URL http://www.digra.org/dl/display_html?chid=06276.09313.pdf. Accessed on 18.6.2007.