A game generator -
the framework for
an educational
system
development game

Master thesis

Anne Chr. S. Landro

1st August 2007
A game generator - the framework for an educational system development game

Anne Chr. S. Landro

1st August 2007
Abstract

Games have an engaging nature, and educational games can help make the learning process more interesting. At present, there is no Norwegian educational game that simulates the system development process. This master's thesis explores the technical aspects of creating such a game through the development of a prototype game generator.

The main goals of the software were for it to be user friendly, extendable and reusable. Another important aspect was for the games produced by the game generator to be appealing to play. Such a product can help transfer the fun aspect of games to education.

In this system, a game consists of assignments. Each assignment is a task the player must solve. The player's actions should, in part, determine how the game unfolds. Thus, the concept of handlers was introduced. They determine what is the next assignment based on the player's answer to the current assignment.

The handlers are separate from the rest of the code, thus they are easily extended, added to, or changed. To demonstrate the functionality of the concept, several handler types were developed. The most noteworthy of these is the custom handler, which allows the game author to make his own handler. Consequently, he can extend the system without changing the code.

The system is data-driven. The logic resides in the data, rather than in the system. Hence, by changing the data, you change the game. The entire system is built upon the idea of separating logic from content.

A qualitative test was conducted to see if the software was usable and user friendly. The results were promising, and revealed that it is relatively easy to create a system development game using the game generator. The resulting game is also very easy to use.
Acknowledgments

First, I would like to thank my supervisors, Gerhard Skagestein, Jens Kaasbøll and Odd Aurmo. I appreciate all the time, effort and expertise they have offered. Without them, I could never have written this thesis.

I would like to thank my father for teaching me how to write, and for helping me with proof reading. To my mother, for showing me how to be independent. To my sister, for always being my pillar of strength. She is also the artist behind the avatar images in this thesis. To Marius, for making me laugh, and re-introducing my to my playful side. To Nils and Rustam for commenting on my work.

Last, but not least, I would like to thank all my family and friends for their never-ending support, and for always believing in me.
Contents

1 Introduction ......................................................................................... 1
  1.1 Background ............................................................................. 1
  1.2 Contribution ........................................................................... 2
    1.2.1 The users ........................................................................ 3
  1.3 Outline of the thesis ................................................................. 3

2 Background ...................................................................................... 6
  2.1 The importance of computer games ............................................ 6
  2.2 The value of educational computer games ................................... 10
  2.3 Previous work .......................................................................... 13
  2.4 Summary .................................................................................. 15

3 Designing a game ............................................................................. 16
  3.1 What is a game? ........................................................................ 16
    3.1.1 Rules .............................................................................. 17
    3.1.2 Variable, quantifiable outcome ........................................... 17
    3.1.3 Value assigned to possible outcomes ................................... 18
    3.1.4 Player effort ..................................................................... 18
    3.1.5 Player attached to outcome ............................................... 18
    3.1.6 Negotiable consequences .................................................. 19
  3.2 What makes players play games .................................................. 19
    3.2.1 Challenges ....................................................................... 20
    3.2.2 Socialize .......................................................................... 21
    3.2.3 Dynamic solitary experience ............................................. 21
    3.2.4 Bragging rights .................................................................. 21
    3.2.5 Emotional experience ....................................................... 21
3.2.6 Explore ................................................................. 22
3.2.7 Fantasize .............................................................. 22
3.2.8 Interactivity ......................................................... 22
3.2.9 Getting better at something .................................... 23
3.2.10 Clear and compelling goals .................................... 23
3.2.11 Identification with main character ........................... 24
3.3 Making an educational game ....................................... 25
3.3.1 Providing feedback ................................................ 25
3.4 Summary ................................................................. 26

4 Specifications .................................................................. 27
4.1 System design characteristics ....................................... 27
4.1.1 User friendly .......................................................... 27
4.1.2 Extendable ............................................................. 28
4.1.3 Reusable ............................................................... 28
4.2 Functional requirements ............................................... 29
4.2.1 The game generator ................................................ 29
4.2.2 The game .............................................................. 30
4.3 Summary ................................................................. 31

5 Technologies .................................................................. 32
5.1 Programming language ............................................... 32
5.2 Web technologies ........................................................ 34
5.2.1 XML ................................................................. 34
5.2.2 DTD ................................................................. 35
5.2.3 XSLT ................................................................. 36
5.3 Development tools ...................................................... 37
5.3.1 SVN ................................................................. 37
5.3.2 Eclipse ............................................................... 39
5.4 Using existing software ............................................... 40
5.4.1 Lectora ............................................................... 41
5.4.2 Game editors ......................................................... 43
5.4.3 Conclusion .......................................................... 45
5.5 Summary ................................................................. 45
6 Design

6.1 Inside the game .............................................. 47
  6.1.1 Scoring .............................................. 47
  6.1.2 Assignments ........................................... 48
  6.1.3 The flow of the game ................................. 49
  6.1.4 Handlers ............................................. 50
  6.1.5 Minimal handler ....................................... 51
  6.1.6 Random handler ...................................... 52
  6.1.7 Choice .............................................. 52
  6.1.8 Number Compare ..................................... 52
  6.1.9 Question Handler ..................................... 53
  6.1.10 Custom Handler ...................................... 53

6.2 Data-driven system ........................................... 55
  6.2.1 A state machine perspective ......................... 55

6.3 Summary ..................................................... 56

7 Implementation ................................................. 57

7.1 Overview .................................................. 57
  7.1.1 Walkthrough from the player's perspective ........ 57
  7.1.2 The making of a system development game .......... 59
  7.1.3 Class diagram ........................................ 63

7.2 The MVC model ............................................... 63

7.3 Model .................................................... 65
  7.3.1 Game ................................................. 65
  7.3.2 Assignments .......................................... 66
  7.3.3 Random ............................................... 67
  7.3.4 Handler ............................................... 67
  7.3.5 Outcome ............................................. 69
  7.3.6 The relationship between Assignment and Handler .. 69
  7.3.7 Player ............................................... 70
  7.3.8 Assignment List ....................................... 71
  7.3.9 Assignment Info ...................................... 71
  7.3.10 Game List ........................................... 71
  7.3.11 Game Info ........................................... 71
  7.3.12 File Info ........................................... 72
7.4 Controller .................................................. 72
  7.4.1 The game (sysgame) .................. 72
  7.4.2 The game generator (creator) .... 73
7.5 View .................................................. 73
7.6 File system ........................................... 75
  7.6.1 XML-files versus database storage .. 75
  7.6.2 The directory structure ............... 76
7.7 Summary ............................................... 82

8 Testing .................................................. 86
  8.1 Goal .................................................. 86
  8.2 Evaluation technique ....................... 87
    8.2.1 Think aloud evaluation .......... 87
  8.3 How the test was conducted ............. 88
    8.3.1 The participant .................. 88
    8.3.2 The conditions ............ 88
    8.3.3 The tasks ................ 89
  8.4 Results ............................................ 89
    8.4.1 The user manual .......... 89
    8.4.2 The game creator ........ 90
    8.4.3 The game .................. 92
    8.4.4 The interview .......... 92
  8.5 Conclusions .................................... 93
  8.6 Summary .................................... 94

9 Discussion ........................................... 95
  9.1 Does the software meet the specifications? 95
    9.1.1 System design characteristics .... 95
    9.1.2 Functional requirements ........ 100
  9.2 Is the game appealing for the players? 102
    9.2.1 Challenges ................ 102
    9.2.2 Socialize ................ 104
    9.2.3 Dynamic Solitary Experience ... 104
    9.2.4 Bragging rights ........ 104
    9.2.5 Emotional experience ...... 105
9.2.6 Explore ................................................................. 105
9.2.7 Fantasize ............................................................. 106
9.2.8 Interactivity ......................................................... 106
9.2.9 Getting better at something ................................. 107
9.2.10 Clear and compelling goals ......................... 108
9.2.11 Identification with the main character .......... 108
9.2.12 Providing feedback ........................................... 109
9.3 What could have been done differently? ............... 110
9.4 Conclusion ............................................................ 111
9.5 Summary ............................................................... 112

10 Future work .......................................................... 113
10.1 Important improvements .................................... 113
10.1.1 GUI ................................................................. 113
10.1.2 Handlers .......................................................... 117
10.1.3 Graphics and sound ......................................... 118
10.1.4 Making the game a more social experience .... 121
10.1.5 Developing a storyline for the game .......... 121
10.2 Other uses ......................................................... 122
10.2.1 Reusing the game content ......................... 122
10.2.2 Different types of games ............................... 123
10.3 Summary ............................................................. 123

11 Conclusion ............................................................ 125

References ................................................................. 128

A Contents of the CD .................................................. 133
A.1 thesis ................................................................. 133
A.2 src ................................................................. 133
A.3 doc ................................................................. 133
A.4 test ................................................................. 133

B User manual ........................................................... 134
B.1 The structure of the game ................................. 134
B.1.1 Counters ........................................................ 134
List of Figures

2.1 Picture of the offer for 5000 gold ................................. 7
2.2 Ebay entries for WoW guides ........................................ 8
2.3 Prices for WoW gold ................................................ 9

5.1 Screenshot from Lectora ............................................. 42
5.2 Screenshot from Game Editor ....................................... 44

6.1 Example of the flow in a small game ............................... 50
6.2 Relationship between handler, input and outcome ............... 51
6.3 A UML statechart diagram of the system ......................... 56

7.1 Screenshot from the game .......................................... 58
7.2 Making a new game ............................................... 60
7.3 Making a new assignment, part 1 ................................. 61
7.4 Making a minimal handler ......................................... 62
7.5 UML class diagram of the system ................................. 64
7.6 Overview of the game files ......................................... 77
7.7 Overview of the player files ........................................ 82

9.1 Screenshot prior to improvement ................................... 96
9.2 Screenshot after improvement ...................................... 97
9.3 Avatars .................................................................. 109

10.1 A possible GUI version of the game creator ...................... 115
10.2 Talking heads demo .............................................. 119
List of Tables

3.1 Main features of games ............................................. 17
3.2 Reasons players play .............................................. 20

5.1 Example of an XML document ..................................... 35
5.2 Example of an XML document with an internal DTD .............. 37
5.3 Example of an XSLT stylesheet for an XML document ............. 38

7.1 Example of `games.xml` .............................................. 78
7.2 Example of `game.xml` .............................................. 79
7.3 Example of `assignments.xml` ...................................... 80
7.4 Example of `assignment.xml` ....................................... 81
7.5 Example of `players.xml` ........................................... 83
7.6 Example of `player1.xml` ........................................... 84
7.7 Example of `game1.xml` ............................................ 85
Chapter 1

Introduction

1.1 Background

Games have an engaging nature, and educational games seek to take advantage of this in order to make the learning process more interesting. An example of an education that might benefit from this, is system development.

At present, there is no Norwegian educational game that teaches system development principles. This thesis will look at the technical aspects of making a game that simulates a system development process.

The aim is that the finished game will be so close to the reality of the system development process that novice students will gain a useful understanding of what awaits them when working on actual, real-life projects. Furthermore, the game has to be exciting enough to keep the interest of the students. It will consist of different cases typical to system development in Norway.

In such a game, the players can learn to handle different types of assignments, such as text documents, UML-diagram, video messages etc. In addition, random challenges such as resignations, sickness, and computer crashes will be introduced to add realism to the development process. The chain of events will be based on the choices the player makes throughout the game.
CHAPTER 1. INTRODUCTION

To create a system development game, the author needs a technical platform on which to build the game. Available game editors are mainly focused on the graphical part of gaming. For simulation games, though, graphics is not the main concern. Rather, the vital point is the complex logic that drives the game.

The thesis will explore the technical aspects of a system development game through the development of a prototype game generator. This will be a software that allows the game author to build a system development game. The game is made playable by a corresponding game software.

1.2 Contribution

There are two aspects to a system development game. One is the game scenario, that is the storyline, the challenges the player faces. The other is the technical platform that takes the game scenario and transforms it from a story to a game.

In this thesis, I will take a technical perspective on a system development game. This means that I will develop a software to create a game. I will not attempt to make any system development storyline. That task is left to others who specialize in system development.

Because there is no complete system development storylines at the moment, I will make a game generator rather than a game. The game generator will be a software that you can feed a game scenario, and that will create a webgame based on this information.

The system I create will consist of two parts. The first is the game authoring tool. This is the game generator, where you input the storyline, and create the game. The other part is the game itself, where you can play the game you have just made. The idea is that both units will use the same architecture. The game authoring tool builds the game structure, and the game makes use of it afterwards.

My focus is on finding a suitable structure for a system development game and developing a software around this architecture. I will not put effort into developing
effects such as graphics and sound. Instead, I will lay a groundwork that facilitates such additions in the future.

1.2.1 The users

There are primarily three different users that will work with the system. These are the game author, the player, and the programmer. They will each make use of a different part of the system.

The game author
The game author is the person who creates the game. He will be responsible for making the storyline of the game. The part of the system that he will work with, is the game generator. A typical game author will be a professor of a system development course.

The player
The player is the one who plays the game. He will use the game part of the software, along with the result of the game author's work. The standard player is meant to be a student in a system development course.

The programmer
The programmer is the one who makes changes to the software. As what I will make is merely a prototype, it will need to be built upon in the future. The programmer is responsible for making these changes. He will work with the software itself, the part that is hidden from the other users.

1.3 Outline of the thesis

In chapter 2, I look at what an important role computer games play in today’s society. Next, I turn the focus to educational computer games and the research on how they affect learning. I conclude by examining what previous work has been done in the field of system development games, and how my own contribution complements this area.
CHAPTER 1. INTRODUCTION

Chapter 3 concentrates on what a game is. I start by defining the concept of games. Next, I look at what features make games so appealing to the players. The chapter ends with a look at what elements you must take into account when making an educational game.

In chapter 4, I determine the specifications for the system. I start by discussing the system design characteristics. Then I go on to look at the functional requirements of the software.

In chapter 5, I present the main technologies I have used to develop the software. I start by discussing my reasoning for choice of programming language. Then I look at the web technologies I have used. Next, I introduce the main development tools I have employed. I conclude with a discussion of whether to use pre-existing software.

Chapter 6 focuses on the design of the system. I start by explaining the main components of the game. The chapter concludes with a discussion on data-driven code.

Chapter 7 takes a more detailed look at the components of the system. First, I give a brief overview of the system. Then, I present the MVC model. Afterwards, I look at the system divided into the parts of the MVC model - model, view, and controller. The chapter ends with a look at the file system.

In chapter 8, I test the software on a user. First, I outline the main goal I want to attain through the test. Next, I give a quick overview of the technique I use, think aloud evaluation. The subsequent step is to explain how the test was conducted. I then turn to the results of the test. The chapter ends with the conclusions that were drawn from the results.

Chapter 9 examines whether I have managed to achieve my ambitions. First, I look at to what degree the software meets the specifications set forth. Next, I investigate whether the games created by the game generator, is exciting for the players. Then I consider some points that could have been done differently. I end the chapter with a conclusion as to whether I have met my goals.
Chapter 10 focuses on the next steps for this software. First, I address the most important future improvements to the software. To conclude the chapter, I suggest some alternative uses for the system.

Chapter 11 brings the thesis to conclusion. Here, I summarize the main strengths of the software, and discuss whether I have completed my task successfully.
Chapter 2

Background

In this chapter I start by showing what an important part of people's lives computer games play. After having looked at how engaging games in general can be, I will go on to study some examples of educational computer games, and to what degree they have achieved their goals. The chapter ends with an inspection of the previous work with system development games, and how my game generator will enrich and complement the picture.

2.1 The importance of computer games

One of the unique qualities of computer games, is their ability to engage the players. *World of Warcraft* (WoW) (*World of Warcraft* n.d.) is a relatively new game that has taken the gaming world by storm. It is a Massively Multi-player Online Role-Playing Game (MMORPG), which means that it has a vast number of players. *Blizzard*, the developers of WoW, recently announced that WoW has surpassed 9 million subscribers worldwide (*World of Warcraft surpassed 9 million subscribers worldwide Blizzard Entertainment Press Release* n.d.).

Because so many are playing this game, it is easy to find examples of just how important this particular game is in the player's life. One extreme story is the girl who wanted an 'epic mount', a very special horse, for her World of Warcraft character.

Rather than merely attaining the 5000 gold needed to buy the horse through playing World of Warcraft, she decided to buy it, giving the seller a night with her in return.
She posted her picture, and the message 'Hello. I need 5000 world of (sic!) gold for my epic flying mount. In return you can mount me.' on a World of Warcraft message board. Figure 2.1 shows this offer.

![Figure 2.1: Picture of the offer for 5000 gold (Fulgerica n.d.)](image)

The story is, naturally, hard to verify. However, the girl in question was willing to post her picture and evidence of having obtained her epic flying mount. Evidently, she was also pleased with her bargain. These factors add credibility, and suggest that the transaction did in fact take place.

This is, of course, one of the most extreme stories of how much the World of Warcraft game means to its players. While particularly eccentric, this is not the only
example of players who are willing to pay in real-life currency to advance in WoW. Figure 2.2 shows a search for 'WoW' on ebay (ebay n.d.). It reveals that there are several guides for sale, giving instructions on how to improve in the game.

It is also possible to buy gold, items, and even accounts for WoW. WoW gold price list (WoW gold price list n.d.) is one of many pages devoted to WoW. It gives its readers a comparative list of prices for WoW gold, as well as recommendations of where to buy your items or accounts. A high level World of Warcraft character is estimated to be worth about NOK 6000 (Laran n.d.).

Another example of the impact computer games have, comes from the Korean race car game Kart Rider (Kart Rider n.d.). This game has become so popular, that the best players are able to play professionally. They are sponsored by companies, and earn their living by racing virtual cars. Some tournaments in the game are also sponsored, and the prize money has amounted to up to $ 50,000 (Dude, Where’s My Digital Car? n.d.).
CHAPTER 2. BACKGROUND

Figure 2.3: Prices for WoW gold (WoW gold price list n.d.)

What this illustrates, is that the term 'it is only a game' no longer applies to (certain) computer games. When the players are so enraptured in the games that they are willing to spend real-life currency to improve their character, the boundaries between the game world and the real world are growing dimmer.

That computer games are anything but 'just' games, is also apparent for all those who have become seriously addicted. For some, work, school, friends, and family all fade in comparison to WoW. This addiction can even become so serious that it has to be treated in a hospital. It is treated as any other addiction, and playing computer games is compared to using narcotics. (Spillavhengige får behandling n.d.)

In China, computer game addiction is such a big problem that special rehabilitation clinics are dedicated to treat the disease. The government has estimated that about 13 percent of gamers under 18 years are addicted to the Internet (Hektet på nettspill n.d.).
All this gives a clear indication of what an important part of people's lives computer games are. Of course, not all games are equally fetching, and not everyone is affected to the same degree. However, games seem to have a riveting effect on a large group of the population.

I will look closer at some traits that makes games so popular in section 3.2. At this point, it suffices to say that they have a captivating nature. By making educational computer games, you can take advantage of this to capture the interest of the students in a new way.

### 2.2 The value of educational computer games

The previous section illustrated the engaging power of computer games. By taking advantage of these qualities, it is possible to make education easier and more motivating for students.

This idea has sparked an own industry, that focuses on making edutainment concepts. Edutainment is entertainment that is intended to be educational (edutainment n.d.). While edutainment includes such media as television, movies, and books, I will focus on games.

There are several reasons why computer games can be beneficial to education. They are interactive and user-centered, and allow students to 'learn by doing'. However, the main advantage is that they motivate their players, as well as engage them deeply. These are areas where the traditional educational methods fall short (Denis & Jouvelot 2005).

Computer games are indispensable tools when it comes to training in cases where it is too expensive, or even impossible, to acquire the practical experience needed to learn a skill. Examples of such scenarios can be learning to fly a plane or a spacecraft. These are, indeed, areas that have benefited from simulation technology (Solomon 1986).
Another advantage of performing a simulation, is that you avoid the potential harmful effects that may occur. Students can learn from mistakes without having to make them in real life. It is also possible to control what challenges the students will face, targeting the lessons to what is most relevant to learn. In addition, simulations often require less time than real world experience (Baker, Navarro & van der Hoek 2005).

System development is one field that would benefit from using educational simulation games. Software engineering companies are generally not satisfied with the real-world preparedness for university graduates entering the work force. Because the gap between theory and practice is so large, most companies must expect to train the graduates they hire in order to supply this lack of experience. Thus, there is a need to supplement the existing education with more practice that is oriented towards real life. (Baker et al. 2005)

Another, related, point is that research suggests that employees playing computer games at work can be beneficial for the company. A study conducted by the University of Utrecht, Netherlands, allowed a group of employees to play computer games for up to one hour each day at work. This group was more positive to their work than the control group (Games at work may be good for you n.d.).

While the results were related to employees emotions towards their job, it is reasonable to assume that the findings can be transferred to an educational environment. It is worth noting, however, that the computer games played in the study were completely unrelated to the job. This means that the game was a break in the routine, where the employees were allowed to ignore their work for a while.

An educational game would, of course, not offer such a complete break, as the topic would still be the same. Nevertheless, it will constitute a variation from the more traditional learning environment. Also, combining an enjoyable task, such as playing a game, with the more tedious task of studying, will most likely make the study more interesting. Therefore, it is probable that some of the effect noted in the study above will still apply to an educational game. Some research that supports this is discussed later in this section.
The idea of exploiting the driving force computer games have on their players, is not a new one. While gaming is not traditionally considered a physical activity, some companies have sought to make games which challenge this.

*Dance Dance Revolution* (DDR) *(DDR n.d.)* is a series of dancing games originally found in arcades, that has later been adapted to computers and game consoles. By using a 'dancing mat' with arrows, the aim is to dance according to the instructions on the screen.

This type of game has been very popular across the world, and in Norway it has its own club - *DDR Norway*. It arranges dancing competitions, and there is even a Norwegian Cup of DDR.

This is an example where taking advantage of the appeal of computer games helps achieve a 'higher' goal of getting children in better shape. In fact, it is such a success that some American schools have started using *DDR* as part of the physical education curriculum (Schiesel n.d.). In Norway, similar ideas are being tested (Bryne n.d.).

A study of *DEAL*, a business simulation game, suggests that students find such games beneficial to the education. The majority of the participants would recommend business gaming to other students. This is consistent with similar studies.

The students were also asked to determine where computer simulations can be useful. They indicated that it could play an important role in several areas, from improving general problem identification to developing different skills such as decision-making and planning. They also reported that *DEAL* had been helpful in testing understanding on several topics such as 'what is my business' and 'bargaining and negotiating in strategic management', to mention a few.

The business game was reported to have several benefits. Some of the positive effects of playing *DEAL* were understanding the consequences of decision-making, improved understanding of the market mechanism, and that the students were able to integrate knowledge from different subjects.
About half of the participants rated the game as 'fun and interesting'. Most of the students reported that the game was average, or above, on linking the course to reality. In another study, almost half of the participants felt that business strategy games provided a link between theory and practice, by letting them apply what they had learned in a real-life situation (Chang, Lee, ling Ng & Moon 2003).

One study looked at how playing a mathematics game on a handheld game-boy influenced the education for second graders. The experiment revealed that the students completed three times as many math questions using this game than they normally would using a worksheet. In addition, the teachers reported the added benefit that the students helped each other play the game. Playing the game also improved the general classroom culture (Lee, Luchini, Michael, Norris & Soloway 2004).

Another study looked at VR-engage, a game designed to teach geography. The students were given the choice of learning geography through playing the game, or using an educational software application that did not have any game elements, and no virtual reality.

The results revealed that the participants preferred the game to the other software. However, when given the choice to play a commercial game instead, most students with some gaming experience optet for this. This shows that although educational games are more favored than traditional education, there is still a way to go before this type of game will reach the popularity of their commercial counterparts (Virvou, Katsionis & Manos 2004).

2.3 Previous work

One attempt at making a system engineering game, is Problems and Programmers (Problems and Programmers n.d.). It is a card game exploring the possibilities of teaching the process of software development. A test study consisting of students familiar with the theme showed promising results. The subjects reported that they enjoyed the game, and that they thought it a valuable addition to the software engineering curriculum (Baker et al. 2005).
There have also been made several educational computer games aiming to teach the process of system engineering. *SimjavaSP* is a graphical computer game designed to educate the players in system engineering. It allows the player to assume the identity of a project manager. He has to handle a large project by taking charge of the developers, allotting time for different activities, and making purchases, among other responsibilities.

An evaluation of this game revealed that the participants were positive towards it. The majority rated playing the game an enjoyable experience. Likewise, about two thirds signaled that the game taught software development life cycles well. The rest were less satisfied, indicating that it is possible to make further improvements to the game (Shaw & Dermoudy 2005).

*SimjavaSP* is a program made specifically for one game. This means that if you wanted another type of system engineering game, you would most likely have to make extensive changes to the code. This is a drawback, as the game can only be used for one purpose, and might not fit all system development curricula.

Another similar game allows more flexibility in this area. *SimSE* (*SimSE Online* n.d.) is also a system engineering game, and has a lot in common with *SimjavaSP*. They both provide graphical interfaces, and focus on the management side of system engineering. However, *SimSE* is more of a game generator than *SimjavaSP*.

With *SimSE*, the game author has the ability to create the game he wants by deciding which objects and rules govern the game. This makes it possible for the game to change from simulating one system development model to another, for instance. (Navarro & van der Hoek 2004)

Although *SimSE* allows for some freedom in making the game, it is still bound to the management side of the system development process. There are no options that will allow the user to experience actual interaction with customers. Neither is it possible to let the player make actual models of the system, to mention but a few of the opportunities that are lacking.
While both SimSE and SimjavaSP are extensive tools for learning to manage a system development project, they are not useful for teaching the other parts of the engineering process. I have, in fact, been unable to find any games that focus on tasks such as those mentioned in the previous paragraph.

Even if such a game existed, it would be beneficial for professors to be able to make their own games. Then they could concentrate on the points of the system development process that is most important to their particular course. To facilitate this, you need more than just a system development game, you need a game generator.

One software that could have fitted this need, is the Extensible Graphical Game Generator (EGGG). However, this program is designed to make games that follow specific rules. Examples of possible games it can build, are cardgames such as poker, and boardgames such as chess (Orwant 2000). A more complex simulation game without rules to govern the moves, is beyond its capability to create.

As far as I have been able to ascertain, there are no game generators that can make a simulation game complex enough to mimic the process of system development. To fill the need for professors to make games tailored to their courses, I have designed my game generator. It is a software that allows anyone to make a simulation game, with little or no knowledge of programming.

### 2.4 Summary

Computer games are an important part of people’s lives. They can be addictive, and may have an impact on how people behave in the real world. Taking advantage of this, there is more and more edutainment software available. Their motive is to educate while at the same time entertain. Research suggests that they can have a positive influence on education, but that there is still room for improvements.

Some attempts have been made to make system development games. However, these are focused on the management part of the process, largely ignoring the other parts. To be able to target other areas than management, I have developed my game generator.
Chapter 3

Designing a game

In this chapter, I will pinpoint some core qualities of games. Next, I discuss what makes players play them. Lastly, I go into what is important to consider when making an educational game.

3.1 What is a game?

Although *game* is a concept understood by most people, there are different definitions of the word. Juul has looked at several ways to define a game, and has comprised them in the following definition.

*A game is a rule-based formal system with a variable and quantifiable outcome, where different outcomes are assigned different values, the player exerts effort in order to influence the outcome, the player feels attached to the outcome, and the consequences of the activity are optional and negotiable.* (Juul 2003)

This definition describes 6 main features of games. These are summarized in table 3.1. I will elaborate on each point, and discuss how they relate to a system development game.
3.1.1 Rules

The rules of the game state what the player is allowed to do and how he can win. In classical games like chess, these rules are very explicit. In a simulation game such as a system development game, though, they are less clear. Here, the rules would be what determines how good the player’s actions are, and what happens next.

While the rules of chess are known to the player beforehand, they represent an unknown factor in a simulation game. However, because the game seeks to teach the player about the real world, the consequences should be reasonable and realistic. Although the player might not know what the optimal move is, he should be able to expect that a good move will get him closer to a successful ending.

3.1.2 Variable, quantifiable outcome

In order for a game to be interesting, it needs to have several possible endings. If, regardless of whether you played the game well or bad, you ended up in the same place, you would not be compelled to play.

One of the qualities that make games interesting, is the possibility to influence the outcome. In a system development game, this means that the player's actions must guide where the game is headed. In consequence, there must also be different endings, and several paths to reach the end.
3.1.3 Value assigned to possible outcomes

In addition to several outcomes, there should also be different values associated with different endings. In the simplest games with only two endings, winning is considered good and losing bad. This ensures that the players do their best to win.

In a simulation game, where there is the possibility of more complex outcomes, the endings should be graded in a way that makes it easy to understand whether the player has done well. If all the outcomes are equally good, the player will have no incentive to do his best.

In a system development game, endings can range from doing an excellent job and gaining more customers and money for your company while getting a promotion yourself, to losing clients, or even your job, and damaging your company’s reputation.

To make a game challenging, the successful endings should be harder to reach than the bad ones. This gives the player something to work against, and he will not be as easily bored. The right amount of challenge is important in a game to keep the interest and attention of the player.

3.1.4 Player effort

If a player has to put effort into playing the game, he gets more attached to it. This means that the game must challenge the player, giving him an incentive to do his best.

Another important aspect here, is that the game must be interactive. The player must be able to influence what happens in the game in some way. In a system development game, the player must use the skills he has learned in class in order to complete the game.

3.1.5 Player attached to outcome

If the player is attached to the outcome, he is more likely to play the game. He places an emotional value to a certain outcome. Players will feel happy if they win, and sad
CHAPTER 3. DESIGNING A GAME

if they lose.

This feeling is not always related to effort. Even in games of chance where he has no control over the outcome, the player still wants to win. In a system development simulation, the player should want to see his character succeed.

3.1.6 Negotiable consequences

A game may or may not have consequences in real-life. Playing poker for money is one example where there are real-life consequences, while it is also possible to play the same game just for fun.

The system development game could, if desirable, be designed to have such consequences. How well the students play could for instance influence their grades. However, it is more likely that such a game would have few real-life consequences.

3.2 What makes players play games

In Game design - theory & practice, Richard Rouse II describes some factors that make players want to play games (Rouse III 2005). Another one that has looked at this subject, is Prensky (Prensky 2005). I have consolidated their points in table 3.2.

Some of these are not applicable to a system development game, while others are worth considering. After all, the goal is to make students want to play the game, so it is vital to take into account what motives them to play.

Most of these points are more related to designing the storyline rather than the technical platform. However, it is important to provide for these ideas when building the system, so as not to exclude fundamental aspects of a game and limit the usability of the program.
Why players play games

- Challenges
- Socialize
- Dynamic solitary experience
- Bragging rights
- Emotional experience
- Explore
- Fantasize
- Interactivity
- Getting better at something
- Clear and compelling goals
- Identification with main character

Table 3.2: Reasons players play

3.2.1 Challenges

Rouse argues that providing a challenge is one of the primary motivating factors for single-player games. He also suggests that overcoming a challenge is a learning experience in itself. Because games present a challenge, they engage their players in a different way than more passive media such as books or movies (Rouse III 2005).

Prensky also mentions challenge as an important part of a game. He goes further by saying that it is important to attain the optimal level of challenge. This level will be different for different people (Prensky 2005).

In order for a game to be interesting, it needs to provide the right level of challenge. If the goal is too easy to reach, the player soon loses interest. Likewise, if it is too hard, the player is likely to give up. The 'perfect' game will be able to adapt to each player's skills, providing him with the optimal challenge for his particular qualifications.
CHAPTER 3. DESIGNING A GAME

3.2.2 Socialize

One of the main reasons people play non-computer games such as board or card games, is to socialize with others. The same desire applies to a certain degree to computer games as well. This is the reason why there are so many multi-player computer games in different genres ranging from death-matches to Massively Multi-player Online Role-Playing Games (Rouse III 2005).

3.2.3 Dynamic solitary experience

This point complements the previous one. Some players play to socialize, while others prefer a solitary experience. Games provide a more dynamic experience than other activities. The players are actively engaged, and contribute to the development of the storyline (Rouse III 2005).

3.2.4 Bragging rights

Doing well in games is a way to gain respect from others. This is particularly evident in multi-player death-matches, where players brag when they frag (kill) each other. However, it is also important in single-player games. Here, you have high-score lists where you can show the world how good you are.

It is also common to compare how fast you have finished a game, or a level. Gaming forums is another place where it is obvious that telling others how good a player you are, is an important part of the gaming experience (Rouse III 2005).

3.2.5 Emotional experience

Players seek an emotional payoff when playing a computer game. They identify with their character, and want him to succeed. Certain genres are also associated with different feelings. Quake and Unreal give an adrenaline rush, while Silent Hill invoke a more creepy, scary feeling.

Games are able to provoke feelings because they involve the player. He can influence how the game unfolds through his actions. Because he has almost sole respons-
ibility for whether the game ends well, there are a lot of emotions tied to doing well (Rouse III 2005).

### 3.2.6 Explore

The desire to explore new environments is a major driving force in games. This is especially important in level-bases games, where the character might not necessarily evolve throughout the game.

However, players long to explore more than just their surroundings. To explore such things as new strategic choices, different types of resources, or new personalities of characters in a game can be just as enticing (Rouse III 2005).

### 3.2.7 Fantasize

Another main reason why players play, is that they want to fantasize. Playing games is a way to escape your mundane life and experience a world different from your own. It also allows you to imagine yourself in another's shoes, and face challenges and adventures that you would not normally partake in.

An important aspect of this, is that games hide some of the boring details of life. Few games require the character to eat or sleep, for instance. Instead, the player can focus on the more exciting parts of another's life and encounters (Rouse III 2005).

### 3.2.8 Interactivity

Interactivity is actually the cornerstone of most of the factors explored above. What makes games stand out from other forms of entertainment such as books or movies, is the interactivity. Players are not merely passive witnesses to a story that unfolds. Rather, they contribute to the storyline, and have responsibility to help make the game interesting by their involvement (Rouse III 2005).

When making a game interactive, it is important to find the right balance between control and freedom. If you constrain the players too much, they might end up feeling
like they are watching a movie rather than participating in a game. Too much freedom, on the other hand, can become equally boring (Swartout & van Lent 2003).

In some games, near-total freedom might actually make the game exciting. In the Sims, for instance, there are few boundaries limiting your choices. However, it is likely that this approach would be less effective in other genres, such as first-person shooters (Rouse III 2005).

### 3.2.9 Getting better at something

One of the reasons why players like to play, is the feeling of getting better at something. This is closely tied to the desire for a challenge mentioned earlier. Getting better through leveling-up (going to the next level) is the one aspect most players mention as a motivational and engaging factor.

Leveling-up is an ingredient a lot of games incorporate. In Role Playing Games (RPG) it is common that the characters start at a low level, and advance to a higher level as the player improves. As an added bonus, most characters get better, or have more options available when you reach a new level.

The motivating factor of leveling-up is clearly seen in such simple games as Tetris. The game itself is very monotonous, but players still love to get to the next level, and get the visual confirmation that they have improved their skills (Prensky 2005).

### 3.2.10 Clear and compelling goals

In order for players to want to play a particular game, it needs to have a goal that they wish to achieve. This means that the goal will have to be appealing enough to make the player internalize it.

A goal like 'learn system development' is not very compelling. However, helping the main character get a better position at work, or ensuring that the company improves, are more enticing goals.
Usually, there are more than one goal in a game. Typically, there are subgoals such as completing a level or finishing a quest. In addition, the main goal, 'beating' the game, lies underneath.

### 3.2.11 Identification with main character

One of the most important aspects of a successful game, is the player's ability to identify with the main character. If he does not like the character, it will be harder to be immersed in the game (Rouse III 2005).

That this is a major point in designing an interesting game, is evident in most commercial games. Nearly all of them give the player the option of adapting the character to his own preferences.

In RPGs such as *World of Warcraft*, this is of utmost importance. In these types of adventures, you spend a lot of time creating your alter ego before starting to play. Here, you are able to determine what race and gender your character has, as well as details such as the color and length of the hair, armor and clothes.

In first-person shooter games such as *Doom* or *Quake*, this is less important. However, these games still allow for some refinement of the character. You are usually able to choose between a limited number of premade heroes, thereby making it easier for the player to identify with his character.

The utilization of character identification to strengthen the user's interest, is not unique to games. The same principle is at work in for instance online communities, where you often can choose your own avatar. An avatar is a graphical image that represents a person, as on the Internet (*avatar* n.d.).

In such communities, the members often put a lot of time and effort into creating the perfect avatar to represent themselves. Here, the avatars help other members gain a better understanding of the person (or how he wants to be portrayed).
Putting an image to a username helps give life to a community, and makes the site more interesting. In the same manner, playing a character with a name and a face will make the outcome of the game seem more important. It also helps the story seem more real, as the player’s actions will have the impact on a life, albeit a fictional one.

3.3 Making an educational game

3.3.1 Providing feedback

In order for the game to be educational, it is vital that the players see what effect their choices have on how the story unfolds. It should be evident whether a decision is good or bad (Baker et al. 2005).

If, after having completed the game, the player is unsure of which of his choices were good, or does not understand how his decisions brought on the particular ending, this is a tremendous flaw. In such a case, the educational value is greatly reduced, as the player has no way of knowing what skills he needs to improve.

However, this does not mean that a player who makes all the best moves, automatically should get the optimal ending. Games often contain an element of randomness, adding surprise and unexpected events to the picture.

This can be a particularly effective instrument in an educational game, teaching the students that sometimes unexpected crises occur no matter how thorough they have been. Yet, the player should still understand when his choices were unwise, and when it was merely unfortunate conditions that resulted in a less than perfect ending.

In a lot of games, there is a reward system that helps show the player that he has done a good job. When you have killed a lot of enemies in first-person shooter games, for instance, you are usually rewarded with new weapons or other useful artifacts. Of course, in these cases it is usually self-evident that you have done well.

It is also customary to provide a visual barometer that indicates how clever the player acts. First-person shooter games usually have some sort of measurement of the
health of the playing character. Normally, this is given as a percentage of a maximum value.

3.4 Summary

A game has several features. Six of the more important ones are rules, several possible outcomes with different values attached to them, that the player must put effort into winning, that he is attached to the outcome, and negotiable real-life consequences.

It is important to understand why players want to play games. The main attractors are:

- Challenges
- Socialize
- Dynamic solitary experience
- Bragging rights
- Emotional experience
- Explore
- Fantasize
- Interactivity
- Getting better at something
- Clear and compelling goals
- Identification with main character

In addition, giving feedback on the player’s choices, helps give the game a more educational value.
Chapter 4

Specifications

In this chapter, I determine the specifications for my software. First, I set up the system design characteristics. Then, I look at the functional requirements for my system.

4.1 System design characteristics

When designing a program, there are a great number of concerns to consider. I have chosen three main issues to focus on. These three areas are, in my opinion, the most important ones for developing a technical platform for a system development game.

4.1.1 User friendly

The system should have a low threshold to use, both for the game developer and the player. If the basics of the software are hard to learn, this can scare the users from working with it. The softer learning curve is mainly achieved through keeping the program simple, with more main components and fewer details to consider. The names and words used should be as easy to grasp and intuitively understandable as possible.

The goal is that the game will be self explanatory to such an extent that it takes no explanation to to play the game. I have concluded, however, that it would be unrealistic to set such high standards for the game development software.
In order to make a game, the author needs to understand the main components and structure of which a game consists. He needs to know what limitations the game has, as well as his options. Hence, he should expect to spend some time to familiarize himself with the main concepts.

Consequently, the game generator will need a user manual describing how to create a game. However, once this material is learned, the software should be relatively easy to use.

4.1.2 Extendable

Because this is the first prototype, the program will presumably be extended in the future. This job will most likely be done by someone other than myself. Consequently, it is vital that the code is easy to understand, change and extend as required. Otherwise, a lot of time will be wasted trying to understand the program.

This is accomplished by writing code according to code standards, and by extensive documentation. Additionally, separating the program into smaller units help ease future changes.

4.1.3 Reusable

When I started the work on my thesis, there was only a small, rudimentary game story available to work on. As a consequence, when I have investigated how to build a technical platform for a system development game, it has been on a general basis. Since the software is intended to be able to form the base of several games, it is important that the program is not designed to suit one specific game only.

My object is that the software can be used to develop several different types of system development games. However, it should also be able to make other types of games as well. The intention is to allow the game developer as much freedom as possible, without making the system too complex.
The main approach is to consider what different games have in common, in order to find building blocks fit to apply to various genres. Also, it is important not to limit the game author too much with regard to how the game should progress. Giving him more freedom, makes it easier to produce different types of games.

4.2 Functional requirements

4.2.1 The game generator

The intention
The software should allow the game author to be able to create different games. Consequently, the game should be data-driven, meaning that the logic resides in the data rather than in the program itself. By supplying the game generator with different data, the resulting games will differ from each other. I discuss the advantages of data-driven systems further in section 6.2.

Type of game
The game generator should be able to make system development simulation games. The requirements for such a game is provided in section 4.2.2. The game will consists of assignments - tasks that the player will have to complete to move on to the next part.

User interface
The game author should be able to create the games online through a web interface. This allows the author to use the software regardless of where he is, as long as he is connected to the Internet.

Changing the game
The author can make changes to the game after he has made it. He can add assignments, and connect them to the rest of the game. This should be enabled through the same user interface that creates games.
4.2.2 The game

Type of game
The game should be of the simulation kind, in which the player is faced with different challenges. His responses to each of them should in part determine what challenge he faces next. The game will be a one-player game.

User interface
The user interface should be mainly text based. The game will be available online, through a web interface. This allows the players to play wherever they are, as long as they have an Internet connection.

User input
The player should be able to input different types of answers to the challenges. Examples of such input types can be text, digits, or files.

Recording the player’s answers
All the player’s answers should be recorded and logged. Hence, the user must log onto the system with a username.

Resumable
The player should be able to quit the game at any time. The game should then be saved, so that the play will resume at the same place when the player loads the game anew.

Scoring
The game will contain a score-rating that allows the user to see how well he is doing at all times.
4.3 Summary

In this chapter, I have outlined the specifications for my system. The main system design characteristics, are that the software should be user friendly, extendable and reusable. I also determined the functional requirements for both the game generator and the game.
Chapter 5

Technologies

In this chapter, I give a short overview of the main technologies I have used when developing my game generator. I start out by explaining what considerations I took into account when choosing the programming language.

Next, I present the most important web technologies, XML, DTD and XSLT. Then I take a look at the development tools I have found advantageous to use. To conclude the chapter, I discuss whether it is better to build on existing software, or to make the entire project from scratch.

5.1 Programming language

When starting on a project, you need to decide which programming language is best suited to your particular needs. The game generator is a prototype meant to be extended in the future. Because of this, it was important that the language was easy to understand and learn for future programmers. This made it natural to look at the scripting languages.

Scripting languages can be either server-side or client-side. The first means that the scripts run on the server, as opposed to being run by the client’s browser, as is the case with the latter technology.

Server-side scripts require less of the user in terms of installed software, which is an advantage. This prevents the user from having to download new software to be
able to use the program.

In addition, this puts less strain on the user’s computer. The code is executed on the server and translated into HTML before it is sent to the client. Regardless of what the script requires of work, all the client needs to do, is receive and output the HTML document. Because of these advantages, I chose to use a server-side scripting language.

Because of the limited time available for writing a master’s thesis, I found it most efficient to choose a scripting language that I was already familiar with. This limited the choice to Python (Python n.d.) or PHP (PHP n.d.).

At a first glance, PHP seemed like the most obvious choice. After all, this language was specifically designed for web development. Python, on the other hand, must import several different modules if it is to be used to make webpages.

While the two languages share many features, the main advantage of Python is that it is easier to read the code and learn to use it. As mentioned earlier, that is very important in this particular project, as the system is meant to be extended at a later stage.

Another benefit of Python, is that its ability to do more than PHP. As PHP was meant to be used in web development, its primary features are tied to this area. Python is a more general-purpose scripting language, and has consequently more to offer in other fields.

Although the game generator is currently a web application, the most important part of the program is not web-related. Communicating with a user is just a minor piece of the puzzle. Most of the work happens behind the curtain, with the datastructures and determining the path of the game at the core.

I find that Python is more appropriate for this particular project. This is mainly based on its versatility, and the fact that it is an extremely userfriendly language. It
is better equipped to manage complex tasks than PHP. This is required for analyzing the player’s input.

In addition, Python is a relatively fast language. It is also easy to integrate a Python script with other languages, such as C, C++ and .NET. This is advantageous, as it is possible for programmers to utilize other languages should they see the need to do so.

Another benefit of Python, is that it works on all platforms. This means that the same script can be used regardless of operating system. There is no need to change the code even if you change environment (Lutz & Ascher 1999).

### 5.2 Web technologies

#### 5.2.1 XML

XML is an abbreviation for EXtensible Markup Language, and is designed as a way to structure and describe data. A markup language combines text with tags. These tags give extra information such as what the text describes, or how it should be presented. What makes XML extensible, is that the users are allowed to define their own tags.

As XML is meant to merely describe data, the tags do not determine how the information is presented. One of the advantages of this language, is that it separates data from its presentation. XML is often used in conjunction with style sheet languages such as XSLT. This combination allows you to determine how the information is viewed as well.

Instead of determining how the data will appear to the user, the XML tags give information on what the data contains. Table 5.1 illustrates this.

This example shows one of the strengths of XML. In the first album, the title and the artist name is the same - 'Metallica'. Here, the tags clarify what information the text conveys. This is vital when sending data over the Internet. It simplifies communication, as all parties will know what role the different parts of the message play.
XML document for a CD collection

<music>
  <album>
    <artist>Metallica</artist>
    <title>Metallica</title>
  </album>
  <album>
    <artist>Madonna</artist>
    <title>Like a prayer</title>
  </album>
</music>

Table 5.1: Example of an XML document

It is important to notice that the XML format does not transform the data in any way. It is simply a storage format that describes what information the various parts of the text contain.

The different tags are not determined beforehand, unlike HTML where the user is restricted to choose from a limited number of predefined tags. This means that it is up to the user to choose appropriate and descriptive names for their tags. Such an approach gives the user much responsibility, but also a lot of freedom.

XML became a W3C standard in 1998 (W3Schools n.d.a). The World Wide Web Consortium (Consortium n.d.a) is an international consortium that develops web standards. Their mission is to lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web (Consortium n.d.b).

5.2.2 DTD

A Data Type Definition (DTD) defines what elements a particular XML document is composed of, and where it is correct to place the different elements. It defines the structure of the document by determining what it can contain. A DTD can be thought
of as a 'rule book' of how to compose a particular XML document (W3Schools n.d.b).

DTDs are used to describe the format of XML files. This eases interchange of data, as each user can refer to the DTD to understand the format of the files. In addition, by using the DTD, you ensure that all users follow the same standard when designing documents.

An XML document can either contain an internal DTD, or refer to one outside the document. When using an external DTD, you need to add a line that determines where to locate the DTD, such as this:

```xml
<?xml version="1.0" ?>
<!DOCTYPE music SYSTEM "music.dtd">
```

Table 5.2 gives an example of an XML document using an internal DTD.

### 5.2.3 XSLT

EXtensible Stylesheet Language (XSL) is a style sheet language for XML documents. Because XML says nothing about how the information should appear to the user, it needs an XSL style sheet to determine how the document should be displayed.

XSL Transformation (XSLT) is a language for transforming XML documents. This can be used to create new types of documents, for instance XHTML, from an XML file. These new documents can then be showed in a browser, for instance. XSLT became a W3C recommendation in 1999 (W3Schools n.d.c).

The structure of XSLT has some similarities with XML. They both consist of tags which must be 'opened' and 'closed'. However, there are strict rules as to what elements are allowed in XSLT, and you are not allowed to make your own tags.

Table 5.3 shows one possible XSLT for the example XML document.
An XML document with an internal DTD

```xml
<?xml version="1.0"?>
<!DOCTYPE music [
  <!ELEMENT music (album*)>
  <!ELEMENT album (artist, title)>
  <!ELEMENT artist (#PCDATA)>
  <!ELEMENT title (#PCDATA)>
]>

<music>
  <album>
    <artist>Metallica</artist>
    <title>Metallica</title>
  </album>
  <album>
    <artist>Madonna</artist>
    <title>Like a prayer</title>
  </album>
</music>
```

Table 5.2: Example of an XML document with an internal DTD

## 5.3 Development tools

### 5.3.1 SVN

*Subversion (SVN)* ([SVN](https://subversion.apache.org)) is a version control system. This means that it is a software that manages files and directories, as well as monitor changes made to them. Because SVN stores all versions of the files, it allows you to retrieve older editions of your data, and compare them to your most recent ones.

You create a *repository* for your project. This is where all the files and all their different versions are stored. The repository can be located on a remote server. You check out a local working copy of the project that you make changes to. This way,
CHAPTER 5. TECHNOLOGIES

An XSLT for the example music XML document

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>CD Collection</h2>
        <table>
          <tr>
            <th>Title</th>
            <th>Artist</th>
          </tr>
          <xsl:for-each select="music/album">
            <tr>
              <td><xsl:value-of select="artist"/></td>
              <td><xsl:value-of select="title"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Table 5.3: Example of an XSLT stylesheet for an XML document

SVN can also be used as a complex backup-system.

When you have updated the files, you commit them back into the repository. You can also include a message about what changes have been made. This allows you to get a quick overview of what has been done.

You should only commit correct code into the repository. That way, if you make some changes that are less than optimal, you can always go back to an earlier version. This is very practical if the code suddenly does not work, and you are unsure of what
changes you have made. SVN will allow you to not only revert back to older copies, but also let you compare the two files, and see what is different between them.

Version control systems are invaluable when several people are working on the same project. With SVN, it is possible for different people to make changes to the same file simultaneously, without destroying each other’s work.

However, SVN is a useful tool even if you are a sole programmer. The backup system, and the opportunity of reverting back to older versions, make this a very attractive software (Collins-Sussman, Fitzpatrick & Pilato n.d.)

5.3.2 Eclipse

Eclipse (Eclipse n.d.) is an open source Integrated Development Environment (IDE). An IDE is a type of program designed to ease the development of software. This is typically done through such features as advanced code editors, compilers and debuggers.

Eclipse was originally built to ease Java programming, but through several plug-ins, you are able to enjoy its benefits when programming in other languages as well. Pydev - Python Development Environment - (PyDev n.d.) is one such plug-in that makes it possible to use Eclipse in conjunction with Python and Jython.

The advantages of Eclipse are most evident in larger projects. Its graphical user interface makes it easy to get a quick overview of all the files. This means that you can click on a class or a method to go straight to it, rather than having to search through a potentially large file.

Because Eclipse has built-in compilers and interpreters, it is possible to compile and run your program inside the software. In addition, the debuggers make it easier to find mistakes in the code.

Syntax errors are usually spotted as you write the program, and lines with such problems are marked. Eclipse also provides auto-completion on parts of the code. These features can help reduce the time spent on debugging, thus making the development more efficient.
Eclipse also works well with SVN. Projects are effortlessly imported and exported. Moreover, it provides a useful graphical interface to such important SVN features as comparing a file to an older version. The seamless cooperation with SVN also helps make Eclipse an invaluable tool.

However, there are some drawbacks concerning Pydev. While it mostly provides a good Python development environment, a couple of problems arise. The main issue, is that errors occur when formatting the code. When using a format-sensitive language such as Python, this is a major complication.

Furthermore, because Pydev shows the indentation differently than it is stored in the file, formatting errors can be near impossible to discover. Yet, despite some problems regarding Python, Eclipse has proved to be a valuable tool when developing the system development game.

5.4 Using existing software

When designing a software, you need to consider whether to start from scratch or use pre-existing software. To find out if I should take advantage of existing software, I looked at some of the benefits and drawbacks of using such programs. In addition, I evaluated some of the available software in relation to my particular need.

It can be attractive to buy existing software to build upon rather than start from scratch. The most obvious advantage is that a lot of the work has already been done. This enables you to focus on the important parts of the development, those that set your project apart from others.

Another appealing trait with such software, is that it is often developed by a team of professionals, and sold to several customers. This allows the developers to spend much time, money, and workforce on the program. It would take a lot of resources to duplicate this work, and, depending on your situation, this might not always be feasible.
On the other hand, buying a license to a program can potentially be expensive. Therefore, it is important to consider if the cost is equivalent to what you pay for. Often, when purchasing such a package, it includes advanced features that might be superfluent in your particular project. Yet, you usually have to pay for the entire program, heightening the cost of the items you actually need.

If you choose to buy a commercial software package, you also run the risk of the company discontinuing that particular product, or ceases to exist altogether. This can mean that you will get no support on the program, and might make it harder to extend or change your own code in the future.

Another problem with using existing software, is that it is not adapted to your particular needs. You frequently end up having to customize the software to your program. When designing your own software from scratch, you avoid this issue, as it is made with your specifications in mind.

Making changes and additions to commercial software can be problematic. Often you don’t own the rights to change the code itself, even if you took the time to familiarize yourself with it. This means that you need to build patches based on the existing code, often adding complexity.

Some of the concerns mentioned above can be avoided by using open source rather than commercial software. This will, in turn, have its own advantages and disadvantages. Another solution is to buy a software package and pay extra to get a version that is customized to your needs.

In the end, it comes down to how much time, money, and resources you have available for your particular project, as well as what software is developed in that specific area. I looked at some of the available software on game and web development, and will discuss them below.

5.4.1 Lectora

Lectora (Lectora n.d.) is an e-learning development tool produced by Riventi (Riventi n.d.a), a Norwegian company specializing in authoring tools, knowledge management
solutions, and general e-learning services.

Lectora is an advanced webpage editor specifically designed to create e-learning systems. It offers more complex features than other, more general webpage editors, such as for instance Dreamweaver (Dreamweaver n.d.). It has also won several awards (Nyheter fra Riventi n.d.).

The features include different templates for webpages, with a high degree of adaptability. The user can easily create the type of page he wants with little to no prior knowledge of web programming. In addition, it is a simple task to create tests and surveys. Figure 5.1 gives an impression of the magnitude of the program.

![Screenshot from Lectora](image)

Figure 5.1: Screenshot from Lectora (Riventi n.d.b)

However, there is no support for more advanced logic behind the webpages. In a game, there needs to be some sort of engine that determines what page is next based on the player's input. This is not an option in Lectora.
Presumably, it would be feasible to use Lectora to produce the webpages themselves, and connect them to a different program that determines the path through the game. Nevertheless, this way you would not take advantage of all the product has to offer. It would be equally effective to use any other web editor.

While Lectora seems a favorable option for creating an e-learning environment like a webcourse, it is less useful when making a game. For this purpose, a tool that is more geared towards gaming is needed.

5.4.2 Game editors

There are several game editors that allow the users to make their own 2D or 3D games. Game Editor (editor n.d.) is one such engine.

Game Editor lets the user compose a two-dimensional game world. It also enables the creation of antagonists - creatures or things that the player must battle. In addition, it is possible to specify actions for different events, for instance that the character dies if it collides with an enemy. This is illustrated in figure 5.2.

The editor is cross-platform, so it is possible to publish the games on different operating systems, and even cellular phones.

Although this is a software designed to make games, it is not suitable for making a system development game. Admittedly, the program allows you to connect an event with an action, still this option is too limited for my purpose.

The program is designed to react if the character meets a monster, collides with an object, or reaches a certain point. It is not possible to do advanced actions such as comparing two UML diagrams. Consequently, it is impossible to determine what happens next based on the user input.

It is interesting that the main reason Game Editor is not suitable for making a system development game, is the same as Lectora’s - the lack of support for advanced handling of user input. This is common for most game editors, as they focus more on battling monsters.
The main objective of Game Editor and similar editors, is to provide the users with a tool to easily create a game world and its inhabitants. However, for a system development game, the primary intention is not the graphics. Rather, it is important that the player’s actions get an appropriate response.

Graphics are an important part of games, and in the future it would be interesting to add more life to the system development game by creating a game world. However, at present this is a minor issue. This type of game editors is consequently less relevant for my particular project.
5.4.3 Conclusion

I have been unable to find any existing software that suits my purpose. While some fulfill parts of the requirements, none addresses the major challenge - to interpret the player’s answers and use them to determine the path through the game.

I could have used some of these softwares as a base for my project. However, the work needed to adapt the programs to my needs would likely outweigh the advantages of using them. In addition, there are the other issues with using premade software mentioned earlier. In the end, I decided to start from scratch. This allows me much more freedom to do exactly what I want to do, and I am in complete control of the entire code.

5.5 Summary

It is crucial to choose the perfect programming language for your particular project. I decided on Python, because it is versatile and very easy to understand and learn. In addition, it has adequate support for webprogramming to suit my needs.

The most important web technologies for this particular project, are XML, DTD and XSLT. XML is a markup language used to describe data. With DTDs, you can specify the structure of XML documents. By adding an XSLT stylesheet, you can display the XML documents on different media such as in a browser, on a cellular phone and so on.

When working on a large project, there are several development tools that can make the production go smoother. SVN is a version control system that manages files and monitors what changes you make. Eclipse is an integrated development environment that provide different tools to ease the programming.

One of the most essential decisions is whether to use pre-existing software as a base for your project, or build everything yourself. What approach is best, depends on the type of project, your resources, and what software is available.
For my purposes, it was natural to look at educational development software and game editors. Lectora is an e-learning authoring tool. Game Editor is one of many game editors available. Neither meet my needs, and consequently I decided to build everything from scratch.
Chapter 6

Design

In this chapter, I outline the main design of my system. First, I look at what components a game consists of. Next, I study some of the advantages of a data-driven system.

6.1 Inside the game

6.1.1 Scoring

Three factors indicate how the player is doing in the game. These are time, money, and score.

Time
Time is a measure of how much virtual time the player has to complete the game. It does not say anything about how much actual, real-life time the player uses. Rather, it determines how much time the main player character has left to complete his project.

Money
Money indicates how much virtual money the player has to spend. This can for instance be the amount of money the character's company will spend on this particular project.

Score
Score is a direct barometer of how well the player is doing. If his actions are wise,
the score rises. On the other hand, if he makes unwise decisions, the score falls. The 
score is designed to make sure the player knows how well he is doing at all times.

6.1.2 Assignments

In order to make the game generator generic, it is necessary not to consider each 
game as a whole. I needed to divide the gameplay into smaller parts, and look at the 
relationships between them. It was important to find out whether all games had a 
common structure, and whether this architecture could form the groundwork for a 
system development game.

To find out what the main components of games are, I studied books on how to 
develop games. Unfortunately, they focus mainly on the graphical aspect. Little is 
written on constructing the components in a simulation game.

Instead, I turned to commercial games for inspiration, especially roleplaying games. 
In RPGs, such as World of Warcraft, for instance, you play a character in a fictional 
world. You usually have the opportunity to choose where you want to go. What 
happens in the game depends upon what your character decides to do.

When we break down this virtual world, it contains a collection of adventures the 
player can choose from. In the same way, I have decided to look at a system develop­
ment game as a gathering of assignments.

An assignment can be thought of as a specific event. In an RPG, it might be an 
encounter of some sorts, a fight, or a task to be carried out, to mention a few pos­
sibilities. It is an episode where the player accomplishes something, and it might 
influence the rest of the game.

In a system development game, an assignment can be one of two type - regular or 
random. In a regular assignment, the player is given a specific task to complete, such 
as writing an application.
A variant of this, is a purely informative assignment, in which information is given without requiring any action on the player’s part. The game author is able to add files to the assignment. Examples of this could be a video, an audio clip, pictures or UML diagrams.

The other kind is a random encounter. It is not part of the preplanned plotline, but can happen at almost any time. Which random assignments happen, and when, will consequently vary from game to game.

6.1.3 The flow of the game

The player is given one task, or assignment, at a time. When this is completed, he moves on to the next. One such event can be making a UML diagram, going to a meeting, or watching a video. How he chooses to perform this activity, will determine where he goes next. In this type of game, the road is much more paved ahead of time than in RPGs. Still, the player’s choices will affect how the path of the game unfolds.

I have chosen to represent this connection of assignments as a directed graph. This gives the game developer total freedom in how to draw up a storyline. A figure of how a small sample game might be depicted, is seen in figure 6.1.

As is apparent from the figure, the path from start to goal is not determined ahead of time. Rather, there are several ways to play the game, and different possible endings. We see that one assignment may have various descendants. It is even possible that the path leads back to an earlier assignment.

To support this structure, there needs to be something that differentiates between various user inputs, and decides what assignment comes next. In most games where you have the storyline ahead of time, this can be hard-coded. This gives the developers opportunity to tailor the code specifically to the game story.

Because my game generator should be able to make different games, it needs to be more general. At the same time, it has to be flexible enough to give room for creating interesting stories. It is not sufficient to have only one path through the game. That would give the player no control over the game, which would lower the realism and, in consequence, the learning value.
6.1.4 Handlers

To manage the user input, and determine the next assignment, I have made *handlers*. This is a unified way of interpreting the player’s answers, regardless of the type of assignment.

Based on the input, the handler determines the outcome of the assignment. An outcome consists of how much time the player has used (*time*), how much money he has spent (*money*), how wise his decision was (*score*), and what assignment comes next (*next*). Figure 6.2 depicts how the handler determines the outcome based on the user input.

As can be seen in the figure, this is a kind of *black box* approach to the problem. The rest of the system does not need to be aware of how the handler works. It does not even have to know what type of input it requires. Its only job is to pass on the input, and receive the outcome. The handler has full responsibility for managing the user input and determining the outcome based on this.
Correspondingly, the handler is not aware of what goes on in the rest of the program. The only correspondence between the two parts is input and outcome. As long as this is standardized, the different subsystems can be virtually unaware of each other.

The main advantage of this approach, is that it is easy to extend the system. If you want to introduce a new type of user input/output combination, all that needs to be changed is the handler. And there is almost no need to change the existing code; rather, one would extend it with the new handler.

Only a few minimal additions are needed for the rest of the program. The parts that need to be changed are also kept separate from the remaining system. This limits the number of files being altered to a minimum, making updates easier.

To test out the handler concept in the prototype, I have developed a couple of preliminary handlers. They are designed to show some of the possibilities of the handlers, and to give a foundation upon which to try out the game generator.

### 6.1.5 Minimal handler

This handler is designed to manage those cases where the outcome is the same regardless of what the user does. An example of where this might be useful, is a pure ‘information’ assignment, such as watching a video. The purpose here is to inform the user, and there is no need for him to reply in any way.

Another use for this handler, is if the user input does not influence the next step, but will be used later on. One such assignment might ask the user to take some notes.
These notes are not important at the time, but will be taken up at a later stage in the game. So it is possible to combine this minimal handler with user input, but it will not have consequences for the flow of the game.

It might seem an excessively complex solution to use a handler in this case. After all, the next assignment is set in stone. However, by managing all assignment input, no matter how small and insignificant, through handlers, the program is more consistent. This simplifies the program, and makes maintenance and changes easier.

6.1.6 Random handler

This handler is designed to manage random events. It is quite simple, since the user is unable to influence these events. Consequently, there is no user input to consider. Also, the random events do not determine what assignment comes next. This is the job of the previous assignment's handler.

6.1.7 Choice

Choice handlers are applied when the user is presented with a list of possible answers, and has to choose one of them. An example of this might be that the player is asked which of several UML diagrams he thinks is right. There is also an other option, in which the user can specify his own answer.

6.1.8 Number Compare

This type of handler is used when the user input is a number. It has the option of specifying different outcomes based upon whether the player's answer is lower, higher or equal to a predetermined number.

One way this might be useful, is if the player is asked to estimate the cost of the project. Here, his answer can be compared to a preferred number. If he estimates a too high cost, he might loose the prospect altogether. While a too low number may cause problems later on, if the estimate is unrealistic.
6.1.9 Question Handler

This handler is used when the player is allowed to ask questions. This can be used to simulate a meeting between the player and a potential customer, for instance. The player can then ask for clarifications and clear up misunderstandings and uncertainties.

A file of possible questions and answers are stored in the game. If the user enquires after a topic that is not in this FAQ, an email is sent to the administrator of the game. It is then his responsibility to add an answer to this question. That way the game can grow, and the players actually help develop the game further. This adds realism to the game.

The question handler is in some ways similar to the regular handler. The outcome is already set, and the player's answers do not change this directly. The main responsibility of this handler is to provide the correct answers to the questions the player poses. This makes it differ a little from the rest of the handlers. However, this is just another form of managing the input. So I chose to keep it in a handler, rather than introducing a whole new concept for this.

In addition to maintaining one standard way of managing user input, this handler also demonstrates some of the possibilities that lie in the handler concept. It is not limited to merely determining the outcome of the assignments, but can be made to do additional work such as answering questions. This shows that one of the main strengths of this architecture, is its versatility.

6.1.10 Custom Handler

The custom handler is the most complex of all the handlers, and it allows the user to make his own handler. This is useful if none of the standard handlers applies. To make a custom handler, the author simply makes a small program that takes input and gives an outcome. An example of a trivial custom handler is shown in figure 6.1.10

```bash
#!/store/bin/python
```
def processCustom(input):
    next = None
    time = 0
    money = 0.0
    score = 0.0

    return next, time, money, score

The custom handler enables the user to create his own handler without having to
change the code. This means that no knowledge of the system is needed to extend
or change it to meet a particular demand. If the author have to make changes to the
code itself to create a particular game, it would make the program much less user
friendly, and require more from the author.

In the prototype, the add-on custom handler is made in python. However, it is rel­
atively simple to extend the program to include support for additional programming
languages.

In future editions, it would be beneficial to put as few constra ints on the custom
handler as possible. The opportunity to make the handler in the language of the
author's choice, will allow him to make his own handler without having to learn a
new programming language. At present, however, it suffices to allow only Python
handlers, as the custom handler is meant to purely demonstrate the adaptability of
the system.

I have included the custom handler to make the system as flexible as possible. It
also makes it easier for game authors to make the game they want without too many
limitations. Without a predefined storyline, it is impossible to predict all the types of
handlers needed, and it thus important to include such an option.

If the game generator is too restrictive, it will make the story more artificial and
less realistic. As the game is meant to simulate the real world, this will greatly lessen
both its learning and entertainment value. The custom handler prevents this problem
by opening up the possibilities of the game author. He can add new handler types 'on
the fly’ if the need arises. This also allows the system to grow over time, as the need for new handler types will spark new developments.

Some of these custom handlers will be so universally useful that they might be added as their own handler types. One example of a custom handler that might be used several times over, is a UML interpreter that compares two UML diagrams. Because the handlers are an independent part of the system, this can be done with minimal knowledge of the code.

The development of new handler types will not make the custom handler obsolete, however. Some handlers are so specific to a certain problem, they do not warrant their own handler type. Also, although the system can easily be extended with a new handler, it is even easier to make a custom handler. These reasons make the custom handler an invaluable part of the system.

6.2 Data-driven system

The most important feature of this program, is that it is data-driven. This means that all the logic resides in the data, rather than in the code itself. The advantage of such an approach is that by changing only the data, you get a whole new game.

In traditional games, more of the logic is hard-coded into the program. In order to change the game, you would have to change the program itself. This requires extensive programming skills. When the logic is in the data, however, no knowledge of the code is needed to make changes. This makes it easier for anyone to add to the game, or make a new one.

6.2.1 A state machine perspective

One model that makes use of the benefits of data-driven programming, is a state machine. It is a model of behavior that consists of a finite number of states, the transitions between them, and actions. I have made a simple UML statechart diagram (Figure 6.3) of the system to explain how it can be viewed as a state machine. This diagram only models the main portions of the program.
As can be seen from the diagram, the system has two main tasks - presenting information to the user, and processing the information received from the user. The essential part here is the information. It is not the system itself that decides how the game develops. Depending on what information is fed to the machine, the resulting games will differ from one another.

This works on two levels. One way of modifying the information flow, is to alter the game itself by changing the assignments. This is the job of the game developer. But the user can also influence the game. His answers and choices in the game will determine the path through this specific game. By making other choices, a new path may arise, creating a whole new scenario.

### 6.3 Summary

In this chapter, I have outlined the main design for my system. A game consists of a number of assignments that represent challenges the player faces. Handlers manage the player’s input, and determine what assignment follows the current one.

The software is data-driven rather than code-driven. This means that my system can be compared to a state machine.
Chapter 7

Implementation

In this chapter, I look at how the system works on a detailed level. First, I provide an overview of the system. Then, I introduce the MVC model, and study the different parts of my system in relation to it. To conclude the chapter, I give a brief overview of the file system.

7.1 Overview

7.1.1 Walkthrough from the player’s perspective

Before starting a new game, the player must first log in using his UiO username. By using the name connected to the UiO account, it will be easier to identify the student than if he were to supply a username of his own choice. This will allow course administrators to give feedback on the actions of the player if so desired.

Once he is logged in, the student must choose an avatar to represent him in the game. He can select one of several faces. Both male and female avatars are available.

Next, the student chooses which game he wants to play. The option of selecting the game is added with the thought that later on it might be advantageous to have different games for different parts of the system development process.

When the game is selected, and loaded, the player begins to play. The course of the game is to simulate a system development process. One possible storyline could
start with having to fight for the job and the ending will depend on the actions of the player. He may never get the job, he may spend too much time or money to be able to finish the project, or he may do excellently and improve his firm's clientbase and reputation. Figure 7.1 shows a screenshot from a test game.

![Figure 7.1: Screenshot from the game](image_url)

Along the way, the player is presented with several challenges that he must respond to in different manners, depending on the type of problem. Some parts may not
require any user input at all. This is true for random encounters (both good and bad in nature) and assignments that are pure information.

Other cases will require some sort of choice or action from the player. He will be asked to take notes, ask questions, write proposals and make UML diagrams, for instance. His responses will influence what happens next. Not only will the sum of his actions decide the outcome of the game, it will also determine which challenge he must face next.

The game continues, either until the scenario has reached its conclusion, or the player decides to take a break. In either case, the game, as well as the player's choices along the way, are saved at the start of each new assignment. If the player has chosen to take a break, he is free to resume the play at a later time.

Game over
The game can end in one of two ways. First, the game is over if the storyline is completed. In this case, the player has finished the last assignment of the game scenario. Examples of this, can be that the player loses the contract to another company, or that he has finished the job successfully.

Alternatively, the game can end because one of the three scoring scales - time, money, and score - has dropped too low. If money falls below zero, the player has spent all the money. Likewise, zero time indicates that he has run out of time. A too low score means that the player has made so many unwise decisions that the game is over.

Regardless of why the game ends, the player is always informed as to whether he has done a good job or not. If one of the scoring scales is too low, he is also notified that this is the reason the game ended.

7.1.2 The making of a system development game
The first step in the game creation, is to make the game itself. While most of the information resides in the assignments, a few details are stored about the game itself. Figure 7.2 shows the user interface for this part of the process.
As can be seen in figure 7.2, the author needs to fill in the name, as well as the time and money available at the start of the game. These values will typically decrease as the game unfolds. The random level is a number indicating the percentage chance that the next assignment will be random. As will be explained later, this does not mean that this is an accurate account of the ratio of random encounters in the game. Other factors play a role as well, however, it gives a fair impression.

The score is a measurement of how many bad decisions the player can take before the game is prematurely ended. If the player’s score reaches this level, it means that his actions are so unwise that the project is over. Lastly, the administrators’ email addresses are used to contact them about subjects that may arise during play, that might require changes or additions to a game.

Once the ground rules of the game is set, it is time to add assignments. Figure 7.3 shows the first step in the creation process. The author needs to decide on the name, and what kind of assignment this is. The type is either regular or random assignment.
The author then inputs the content of the assignment. This is a text, and serves two purposes. If can constitute the main part of the assignment. Alternatively, if the main content is the attached file, it can be used as a short introduction to the main part. The last step of this part is to upload the assignment file, if any.

Part two in the creation process is for regular assignments only. At this point, the game authors decides which handler to use, and whether the assignment can be followed by a random encounter or not. Because a random encounter has its own handler, and can never be followed by another random encounter, this step does not apply.
In the last step the author must decide upon one or more outcomes of the assignment. This part depends upon which handler was chosen in the previous step, but figure 7.4 shows the information needed for the simplest handler.

![Figure 7.4: Making a minimal handler](image)

*Next* indicates which assignment will follow this one, *time* and *money* determines how much of both is spent on this assignment, while *score* is a measurement of how good the player's decision was. One outcome consists of these four components. In addition, for this simple handler it is necessary to know what type of input the player might give. This can either be nothing, text, a file or a number.

For the simplest handler, such as the one used for random encounters, this suffices. But for the more complex ones, additional information is required. Choice handlers need one outcome for each possible answer the player can choose between, as well as a list of the available selections. Moreover, an additional outcome is required in case the player chooses the unspecified *other* option. Because the choices are preset, there is no need to specify the input type here.
Likewise, the number compare handler needs three outcomes. One if the player answer is equal to the reference, one if it is less, and one if it is greater than this number. Moreover, the author needs to specify the reference number which the player's answer will be compared to. Again, since the input will always be a number, no input type is specified.

The question handler only has one outcome, but needs some extra information as well. The author needs to upload the file where the questions and answers are stored. Also, he needs to specify from where the possible questions are taken. This is usually information the player has given on a previous assignment. The player's answers will always be questions, so the input type is once more preset.

The custom handler differs a little from the pattern. Because every decision is made in a file uploaded from the author, there is no need to specify any outcomes. However, to be able to make a drawing of the game with all the possible paths, the author must input which possible assignments might come next. Furthermore, he must upload the file containing the special handler, and determine the input type of the assignment.

7.1.3 Class diagram

To give a quick overview of the system, I have developed a UML class diagram (Figure 7.5). It shows the main components along with the relationship between them. I have also indicated what belongs to which parts of the MVC model.

7.2 The MVC model

The MVC model (Model-view-controller) is a design pattern, and the point is to separate different parts of the program in order to ease maintenance and updating. As the name suggests, the system is divided in three - the model, the view and the controller.

The model is the representation of the information in the system. In my program, the model consists of game and player, among other things. The view is the part that presents information to and gets input from the user. In the system development game, this is the parts that makes the webpages. The controller is a coordinator
that responds to changes in the view, and is responsible for invoking updates in the model.

In a typical MVC model, the flow of control is as follows: The view presents information to the user, who then in turn responds. Then the view sends the responses to the controller, which changes the model accordingly. Then the view presents the user with the new information which it gathers from the model.

Figure 7.5: UML class diagram of the system
I have decided to use a passive MVC model. In this design pattern, the controller is responsible for notifying the view on updates in the model. This allows the model to be unaware of the other two parts, which makes reuse easier. Alternatively, I could have chosen an active model in which the model itself notifies other parts about its updates.

The main reason for using the MVC model, is that it eases updating, extending and changing the system. Because the different parts are separated, changes to one part does not affect the others. For instance, it is possible to change the view, without disturbing the other parts. Each component is like a black box, and the others do not need to know what happens within it to communicate properly with it.

The greatest drawback is that the system will be more complex when using this pattern. Another concern is that this division may not be intuitive and easy to achieve. However, as this is a prototype meant to be built upon, it is essential that the system is easy to update. This weighs more than the added complexity.

### 7.3 Model

#### 7.3.1 Game

When a player starts a game, the first thing that happens, is that the game loads. Because of the potential size of the game, only the most basic information is loaded at this point. This includes the total number of assignments and random assignments, contact information for the administrators, and information on where to find the actual assignments and random encounters when it is time to load them.

Another important aspect to know in advance, is how often a random encounter can occur. This random level is a decimal number between 0 and 100. In addition, a **score limit** gives the limit of how bad the overall situation can be (based upon the player’s decisions) before the game is prematurely ended. Each choice will add to or subtract from this score.
When the game is fully loaded, the next step is to play an assignment. At this point, the game decides which assignment comes next. If this is the very first assignment, the choice is simple. If one or more cases are already played, the decision making gets a little more complicated.

First the last assignment needs to be evaluated to see how it affects the game. Depending on the choices of the player, the game may be brought to a conclusion. Alternatively, the answers to the previous assignment can determine which assignment is played next. This is the job of the handler that is connected to the assignment.

Unless the assignment was pure information, involving no choices whatsoever for the player, his moves also needs to be saved. This is both to ensure that this particular game can be reviewed at a later time and because later assignments may depend upon the answers of previous ones.

Next, the program checks to see if the latest played assignment allows for a random encounter. If it does, a random number generator gives a decimal number between 0 and 1, and if this number is below the random level set for the game, the next case will be a random encounter. In that case, another random number is generated, and this is used to determine the identity of the random encounter.

Once the identity of the next assignment is clear, whether a regular or random encounter, it is loaded into the game. This is done based upon the Assignment List datastructures that were loaded at the start of the game. Each assignment identity is linked to a filename, from which each assignment is loaded.

7.3.2 Assignments

This part contains information about the assignments. It contains elements such as assignment name, identity, content and handler. When the assignment is finished loading, it has the responsibility to load the handler as well. It identifies the correct handler type, and loads it.
When an assignment is loaded, and the player has responded to the challenge, the next step is to process this information. The assignment delegates this work to the handler. When it has done its job, there are three variables that describe how wise the player’s choices were — time, money, and score. These are what constitutes the outcome of the assignment.

### 7.3.3 Random

The random assignments are much less complex. They are purely text, and are either good or bad. They can change the player’s available time and money. The overall score is not affected, however, as the random encounters are not linked to the player’s decisions.

A good random assignment is essentially something that positively influences either time or money. Examples of this might be that the client is willing to pay more money, that the company hires more employees, and so on. A bad random encounter is the opposite — it robs the player of time or money. This might for instance be a computer crash, or an unexpected sick leave, or something similar.

### 7.3.4 Handler

This part of the program is responsible for deciding how the player’s actions affect the rest of the game. At this point, there are six different categories of handlers — minimal, random, choice, question handler, number compare, and customized handler.

As with assignments, the handlers are arranged in a hierarchy. The top class, Handler, represents a minimal handler. It is used in the cases where there are no choices available to the player. Such handlers are connected to pure informative assignments, and the next assignment is set in stone, and not dependent upon the actions of the player.

The random handler is equally simple. It is only used in connection with random assignments. This handler only affects time and money. The player’s score is not changed, as he cannot influence the random event. The handler does not determine
what assignment comes next, this is the responsibility of the previous assignment’s handler.

*Choice* handlers are used when there are several options for the player to choose from. He can choose just one, and is limited to the alternatives presented to him from the game. An example of such a handler, would be a case where the answer could be either yes or no, for instance.

The *question handler* comes into use when the player asks questions. His questions are compared to a list of known questions and answers. This is found in a questions file uploaded by the game author. If the question is in the file, the player gets his answer. Otherwise, an email is sent to the administrators of the game, so that they can answer this question at a later time.

The *number compare* handler is used when the player inputs a number, which is then compared to a key solution. It is possible to distinguish between three types of answers - correct, too small and too large. Which category the answer belongs to will then determine the next step in the game.

Of course, such a handler is quite limited as it only allows for three different outcomes. There is at present no way to distinguish between answers that are close to the desired result, and those that are far from it. When this game is developed further, it would be advisable to add such a feature to allow for more adaptability. However, for a mere prototype, this is sufficient to show the possibility of the handler type.

The *custom handler* is by far the most complicated, offering the most flexibility. It allows the game author to write his own code for handling the users answers. This is necessary for the game to be able to handle a vast number of different assignments.

If the 'core code' should be able to manage every possible input, it would be very extensive. Not to mention that it is an impossible task to predict all the types of future assignment. This way, whenever the need for a new type of complex assignment arises, the game author can write a handler specific to this type of problem. This
also facilitates the use of the same handler in more than one game, as it is separately located in a file.

The fact that this handler is so versatile, is one of the reasons why the other handlers are so simple. They are sufficient to demonstrate some basic needs of the game, while more advanced decision making is left to the customized handler. While this suffices for a mere prototype of the system, it is recommended that more handlers are added when extending the program.

### 7.3.5 Outcome

The *Outcome* class contains all information pertaining to the outcome of an assignment. It consists of four parts: *time*, *money*, *score*, and *next*. As previously described, these factors determine how much time has passed, how much money the player has spent, how wise his decisions were, and what assignment should follow the current.

The handler determines the outcome based on the player’s response to the current assignment. It then passes this outcome back to the assignment. Keeping the outcome in a separate class, standardizes the way the handler and assignment communicate. The assignment does not need to know how the handler determines the outcome. Because all outcomes are in the same format, the assignment will always know how to process them.

### 7.3.6 The relationship between Assignment and Handler

To illustrate the relationship between the *assignments* and the *handlers*, figures 7.3.6 and 7.3.6 show what happens when an assignment is played. The figures only show the relevant part of the code, the complete classes are, of course, much more complex.

```python
class Assignment:
    def process(self, input):
        return self.handler.process(input)
```

Example of part of the Assignment class
class Choice(Handler):
    def process(self, input):
        if self.choices.has_key(input):
            self.next = self.choices[input].getNext()
            return self.choices[input]
        else:
            self.next = self.choices["other"].getNext()
            return self.choices["other"]

Example of part of the Choice class

When the player has completed an assignment, his answer is sent to the assignment in question. As can be seen from the figure, the input is passed to the process function. Because the assignment is not responsible for managing the player's input, it merely passes this variable on to its handler. In the example, this is a Choice handler.

The handler, in turn, finds the appropriate outcome based on this input. As is evident from figure 7.3.6, for a Choice handler, the possible outcomes are stored in a dictionary. The handler determines the outcome, and passes it back to the assignment.

The assignment passes the outcome back to the class that called it in the first place. Here, the outcome is stored for further use.

7.3.7 Player

Here all the essential information on each player is stored. In the prototype, this is mainly the username, the avatar image, and a reference to the current game. However, it is easy to extend this part to include for instance sex, age and other information that might be deemed interesting. Although not an option at this point, this can be used to generate statistics as to variations between different groups of players.
7.3.8 Assignment List

Assignment List is a data structure that manages the information on all the assignments - where they are physically located on the disk. It contains a list of Assignment Infos.

This class is used to make lists of all regular and random assignments. It is used to locate the correct Assignment Info for a specific assignment, and keep track of how many assignments the game consists of.

7.3.9 Assignment Info

This is where information on the available assignments for each game is stored. For efficiency, the assignments are loaded on demand, rather than at startup. What is loaded instead, is the basic information on from where to load the different assignments, and this is what the Assignment Info classes contain.

The information these objects hold, is identity, name, assignment type, and filename. Identity is a number used to identify an assignment.

7.3.10 Game List

Game List manages the information on all the games - where they are located on the disk. It consists of a list of Game Infos, and keeps count of how many games there are.

This class makes a list of all the games available. It is used when the player chooses which game to play, or the game author chooses a game to make changes to. This structure allows the software to be used with more than one game.

7.3.11 Game Info

This class contains information on where to find a specific game on the disk. It consists of two parts, the game's name, and the corresponding filename in which all information regarding the game is stored.
7.3.12 File Info

This class holds information regarding where one specific player's current game is stored. It consists of two filenames. One specifies where the player's path through the game, and his answers, are stored. This file is readable by humans, and can be used for later study of how the player played the game.

The other filename refers to where the player’s game is stored. This file is meant for the system only, and allows the game to resume where the player last left it.

7.4 Controller

The controller consists of only two parts, sysgame and creator, and this is where most of the logic behind the game is found. This section is responsible for coordinating information between the view and the model.

7.4.1 The game (sysgame)

Sysgame is the control unit for the game part of the software. When the player starts a game, or input answers to an assignment, this information is passed on to the controller. Based on the variables, sysgame will update the model accordingly.

The first thing to check, is whether the player is logged in or not. If not, the view is called upon to show a login form. Once the player is logged in, he needs to choose an avatar. sysgame passes all the available avatars to the view, and connects the player to his chosen image.

Next, the player will select what game to play. The controller retrieves the information on the available games from a file, and passes this information to the view to present to the user.

Next, the actual game starts. At this point the controller is responsible for deciding whether the play should move to the next assignment or not. In the question assignments, for instance, the player has the option of asking several questions before moving on to the next case.
Unless the game is ended, the controller then asks the view to show the information from the current assignment. If the game has reached a conclusion, the view will be instructed to show the ending text instead. Either way, the avatar image and the player's time, money, and score values are always shown. Sysgame is responsible for gathering the information on the game's status from the model.

7.4.2 The game generator (creator)

This class is responsible for managing the game generator part of the software. First, the game author has to choose between making a new game, or adding to an existing.

If he chooses to create a new game, the creator calls upon the view to show the 'create a game' screen. The game author's input is then passed back to the creator, which in turn creates the game accordingly.

Alternatively, the game author wishes to add to an existing game. Here, the controller loads the list of all the games from the disk, and instructs the view to show this list. The view then passes on information as to which game the author chose, and creator loads this game.

Adding an assignment to a game, consists of three parts - the main assignment information, creating the handler, and determining the outcome. For each section, the controller asks the view to display the corresponding input form. The view passes all the game author's choices back to creator, which in turn creates an assignment based on this.

7.5 View

In this prototype I have decided to make only one view for each part of the system - the game generator and the game itself. The views are responsible for creating the webpages that meet the users. I have not put much emphasis on the aesthetics. The reasoning behind this is that although appearances are an important part of the experience of a game, it is not essential for a prototype.
CHAPTER 7. IMPLEMENTATION

The final user interface should be able to capture the attention of the players. This aspect would be too time-consuming to go sufficiently into in the present thesis. The final edition of this game most definitely needs to address this issue, as discussed in section 10.1.1. However, for my prototype it is of less importance. Consequently, the interface is simplistic, and a rather plain HTML page.

Both views - webcreator and webgame - are almost identical. The first creates webpages for the game generator, while the latter does the same for the game itself. Because these classes are so similar, I considered combining them in one class.

While such an approach could have been feasible in the current version, it would not be an optimal solution. In the future, it is likely that the game's interface will look very different from the game generator's. If they were in the same class, it would be harder to make changes to only one part. Consequently, I decided to keep them separate.

Because all user input is sent through CGI, the first thing the view does, is load all the variables. It is responsible for retrieving the data structures such as game, assignment, and player. When that is done, the view calls upon the controller to do its job.

The view's main job, of course, is to present the information. However, it does have another responsibility as well, to make sure the data are persistent. This is necessary because each time the webpage is reloaded, all the data structures such as game, assignment, and player, are recreated. If these are not saved by the view, the information will be lost.

To survive this transition, all relevant data structures are saved as sessions. Because the Python developers have not made a session module, I have used an open source project called JonPy (Jonpy n.d.) instead.

It is important to note that the view does not in fact save information for later use by other programs. What is saved here, is merely the state of the underlying data structures, so that it doesn’t expire when the page is reloaded. It is not used for any
other purpose. When the user quits the program, the sessions are cleared, to make the computer ready for a new user.

7.6 File system

7.6.1 XML-files versus database storage

When deciding on a storage system for the game, the main options are either to use a database or xml-files. Database storage has several advantages, such as speed and advanced features, yet I prefer the XML approach.

When retrieving specific information, a database is much faster than searching through an XML-document. This is because databases are optimized for information retrieval. In addition, it is also possible to increase search performance further for popular queries.

Databases also have advanced functions such as locks, transactions, indexes, multiuser access, and triggers, to mention a few. However, most of these are not fundamental to this system. Multiuser access, for instance, will not be a problem as each user has his own file, and will never make changes to the game itself.

Only one person can make changes to a game at a time, but this is not perceived as much of an obstacle. Most of the time there will only be one game author per game. In the case of several authors, problems can be avoided through a minimum of coordination.

My main reason for choosing XML based storage is that the format is virtually self explanatory. It takes little time and effort to understand the language, and it is very intuitive. Once the syntax rules are explained, it is relatively easy to design a good XML structure.

Because this is a simple prototype meant to be extended by others, it is essential that the system is easily understood and changed. The use of a database instead, would require prior knowledge of database query languages. Although some
time must be expected for a novice to learn XML as well, the learning curve is much smoother than that of a database query language.

Another point in favor of XML, is that it is universally portable. One can transmit XML electronically without losing any information. This would be hard, if not impossible, to accomplish using a database. In addition, XML can be displayed directly through a web browser. This means that the files can be viewed as they are, without the use of queries to extract information.

The major concern about using XML in this case, is that it works slower than database queries. However, because of the structure of the file system, it is never necessary to extract only parts of the file. As a consequence, the speed problem is reduced. In such a small system, the difference in time will hardly be noticeable for the user.

The security issue is also worth noticing. Databases have the option of more advanced restrictions as to whom can access the files. However, I don't anticipate that this will be a problem. It is possible to apply enough security measures on the XML files to secure that only authorized persons can alter the game. There is no sensitive information that needs to be protected, so the safety measures don't need to be severe.

XML is very portable, and can easily produce different views of the data. Consequently, it would be desirable to make XML files even if a database was chosen as the main storage facility. Using an XML database would only have added to the complexity of the system. The extra features offered are not very useful for this particular project. Ultimately, I decided on XML because it is easier to change, and provides the services needed for this simple system.

### 7.6.2 The directory structure

There are two main directories in the system, *games* and *players*. They are described in the following sections.
CHAPTER 7. IMPLEMENTATION

Game files

Figure 7.6 provides an overview over how the game files are organized. Each game has its own directory, making sure that all files related to a particular game are kept in one place.

Such a separation makes it easier to determine at a glance what information is available for a specific game. It also enables the easy location of the files for that particular game, and provides a logical and intuitive division. A good directory structure is essential to prevent information overload and to ease searching and retrieval.

The root directory contains only one file, games.xml. It contains the name of each available game, and the corresponding filename where the main information of the game is found. Table 7.1 show an example of what this file may look like.
Table 7.1: Example of *games.xml*

<table>
<thead>
<tr>
<th>games.xml</th>
</tr>
</thead>
</table>
| `<?xml version="1.0" ?>
<!DOCTYPE games SYSTEM "games.dtd">
games>
<game>
  <name>A system development game</name>
  <filename>games/game1/game.xml</filename>
</game>
<game>
  <name>A just for fun game</name>
  <filename>games/game2/game.xml</filename>
</game>
</games> |

In each subdirectory *game1*, *game2* etc, there is a file named *game.xml*. It contains the essential information of each game. Here the basic information, such as name and id, is stored, along with where the assignments and random encounters are found. Table 7.2 shows an example of such a file.

Each game directory (*game1*, *game2* etc) contains two more files, *random.xml* and *assignments.xml*. They contain information on what random encounters and assignments are available for the game, and where the information regarding them is found. The two files have an identical structure. Table 7.3 shows an example of an *assignments.xml* file.

As is evident from the diagram, the assignments themselves are all stored in the subdirectory *assignments*. The random encounters are similarly stored in the *random* directory. This is where the actual information on an assignment is found. Table 7.4 shows an example of an assignment.xml file.

Each *game* directory also contains two additional subdirectories, *files* and *players*. The first is the directory in which all uploaded files are stored. This includes for
instance files related to assignments. The players subdirectory in turn, contains all
the system files that saves the games. These are the files that ensures that the player
can resume his game where he left it.

It might seem unnecessarily complex to have so many layers just to reach the ac­
tual assignments. However, with this approach it is easy to retrieve information for
different needs. The file assignments.xml allows us for instance to quickly get an
overview of all assignments in a game. This is important when loading the game, but
it is also convenient for those who wants to chart all assignments without reference
to the content.
The alternative, without this middle layer, would be to go through all the files containing the assignments, and locate the specific information needed in each file. Instead, we are now able to do this in one operation.

A downside to this structure is that there is some data redundancy. An example of this is the assignment type, which is stored both in assignments.xml and the assignment files. Again, this is done in order to ease both the dynamic loading of the assignments, and the reuse of certain files in other connections.

Data redundancy can create consistency problems when updating and changing data. However, as I have created a webinterface to update and change the game, such errors are avoided. The extra time caused by the redundancy when making changes to the game is minimal, and the benefits of the dynamical load possibilities outweigh the disadvantages by far.

**Player files**

All information on the players, reside in the directory players. It contains one file, players.xml, and several subdirectories, one for each player. Figure 7.7 gives an overview of the players directory.
CHAPTER 7. IMPLEMENTATION

assignment.xml

<?xml version="1.0" ?>
<!DOCTYPE assignment SYSTEM "assignment.dtd">
<assignment>
  <name>A job offer</name>
  <id>1</id>
  <type>regular</type>
  <content>You are offered a new job!</content>
  <inputFile></inputFile>
  <random>True</random>
  <handler>
    <type>minimal</type>
    <input>nothing</input>
    <outcome>
      <next>3</next>
      <time>10</time>
      <money>10</money>
      <score>0.2</score>
    </outcome>
  </handler>
</assignment>

<table>
<thead>
<tr>
<th>Table 7.4: Example of assignment.xml</th>
</tr>
</thead>
</table>

players.xml contains information about what players have a profile in the system, and where the data on each one is found. An example of what this file might contain, is found in table 7.5.

Each subdirectory within players, player1, player2 etc, contains detailed information about a player. This is located in the file player1.xml. Table 7.6 gives an example of how this files might look.

The player1 directory also contains a subdirectory called games. It contains xml-files on one or more games the player has played. game1.xml is one such file. Here, all
the choices the player has taken through that particular game is stored. An example on how these files might look, is found in table 7.7.

The *player1* directory also contains another subdirectory, *files*. This is where all the files the player uploads in answer to different assignments, are stored.

### 7.7 Summary

In this section, I have given an outline of how I have implemented my software. First, I looked at what happens when the player plays a game, or the game author creates a game. I then provided a UML class diagram to give a quick overview of the system.

The software follows the MVC model, which divides the system into three parts - the model, the view and the controller. I studied each of these parts in regards to my system.
CHAPTER 7. IMPLEMENTATION

Table 7.5: Example of players.xml

I have used XML files to store the data on the disk. The chapter concludes with an overview of the file system.
### Table 7.6: Example of player1.xml

```xml
<?xml version="1.0" ?>
<!DOCTYPE player SYSTEM "player.dtd">
<player>
  <username>aclandro</username>
  <id>1</id>
  <avatar>images/woman.gif</avatar>
  <games>
    <game>
      <id>1</id>
      <filename>games/players/player1/game1.xml</filename>
      <sysFile>games/game1/players/player1.txt</sysFile>
    </game>
  </games>
</player>
```
Table 7.7: Example of *game1.xml*

```xml
<?xml version="1.0" ?>
<!DOCTYPE games SYSTEM "playerGame.dtd">
<player>
  <id>1</id>
  <game>
    <assignment>
      <id>1</id>
      <type>regular</type>
      <answer>my very best answer</answer>
      <time>50</time>
      <money>500</money>
      <score>0.0</score>
    </assignment>
  </game>
</player>
```
Chapter 8

Testing

In this chapter I test the software on a user. First, I determine what the goal of the testing is. Then I give a brief explanation of the evaluation technique I have chosen, think aloud evaluation.

Next, I outline how the test was conducted. Then I look at the main results of the test. Afterwards, I draw some conclusions based on the test results.

8.1 Goal

The goal of the test is to find out how easy it is to make a system development game using the game generator. It is mainly designed to illuminate the system’s weaknesses, and what parts are not intuitively understood and used.

The test focuses mainly on the interface and functionality of the game generator. A smaller part of the test is also concerned with the game itself, as seen from the player’s perspective.

I have not tested whether the game is educational, or if it can be a beneficial addition to a system development curriculum. The reason for this is that these aspects are tied to the game scenario rather than the technical platform. Also, as there currently is no system development game scenario available, it is difficult to test this now.
The test is meant to highlight the problem areas of the software. This will make it easier to find out what needs to be improved. This knowledge can in turn be used to improve the system in order to make it more user friendly.

### 8.2 Evaluation technique

#### 8.2.1 Think aloud evaluation

Think aloud evaluation is one method of testing the usability of the system. It is a qualitative technique, in that it does not consist of standardized questions. The test is conducted with one test person at a time, with a facilitator that gives the tasks, and one or more observers watching and taking notes.

The user is given a list of tasks relevant for the system that is being tested. He is asked to complete one task after the other while thinking aloud. This means that he will share his thoughts, ideas and reasoning about what he does with the observer.

The method is effective in finding out how easy it is for the user to complete his tasks. In addition, it often illuminates not just what the problem areas are, but also what is the problem is. This is because the user shares his thoughts when the issue arises. Hence, you learn exactly why and how he experiences problems.

When choosing users to test the system, it is important to find participants that are representative of the typical user. Otherwise, the results might be misleading. If a computer novice tests a system that is designed for a computer expert, he will likely experience trouble that may never be an issue for a more advanced user.

One drawback with this evaluation method, is that it may not uncover all, or even the worst, problems in the system. Different people experience different issues, and the results cannot necessarily be generalized. This is a concern with all qualitative tests.

However, the user will likely uncover the most prominent problems with the system. In addition, the test reveals how the user understands the system. This can often be
different from how the developer envisioned, and it is beneficial to learn the user's views.

8.3 How the test was conducted

8.3.1 The participant

The test user was a student taking a master's degree in informatics. He was a man in his twenties.

For this test, there are two user groups - the game authors and the players. A typical game author will be a professor in a system development course. A typical player will be a student in the aforementioned course.

I chose an informatics student because that is close to both the target user groups. He matches the player user group almost perfectly. In addition, it is expected that the game authors have a solid knowledge of informatics, as well as of using the computer and the Internet. The participant matches these prerequisites as well, and was hence a suitable test user.

8.3.2 The conditions

The test was conducted in a quiet environment, with only the test person and myself present. Hence, I filled both the role of facilitator and observer.

I took notes of the most interesting comments from the participant. In addition, the test was recorded on audio. A copy of the sound clip can be found on the CD accompanying this thesis.

The user manual and test game were both written in English. This was not a problem, as the user was fluent in English. The test itself was conducted in Norwegian.
8.3.3 The tasks

First, I explained a little about the procedure to the participant. Next, he was given the assignment of reading the main articles in a couple of newspapers, and looking for a book in an online store. This was to make him comfortable, and let him get used to the technique.

After this initial 'warm-up', the user was asked to read the user manual while commenting on his thoughts about what he read, and any questions that arose. Please refer to appendix B for a copy of the user manual.

When he was satisfied that he understood the basics, he was given a test game scenario. He was instructed to make this game using the game generator. A copy of the test game scenario can be found in appendix C. It is partly based on a preliminary game scenario written by one of my supervisors, Gerhard Skagstein.

I gave the user more hints than is typical for a think-aloud evaluation. This was to ensure that he tried all the parts of the program. However, I did not give hints until after he had made his own choices, and explained the reasoning behind them. Thus, these hints should bear no consequence on the results.

After having completed the game creation, the participant was then asked to play the game. When he had finished the game, the think aloud portion of the test was over. Afterwards I had a qualitative interview with the user to clear up any misunderstandings and ask additional questions.

8.4 Results

8.4.1 The user manual

The test user chose to skim the user manual, rather than study it in detail. He also commented that the average user would not bother to read the manual. Therefore, he advised that all information should be available in the game generator software as well.
The concept of *handlers* was unclear to the participant. However, he reasoned that he would understand them better once he started to make a game. He expressed that it would be different to make a game as compared to reading about it. Consequently, he decided to go on rather than re-read parts of the manual.

Another issue he was initially confused about, was the *next* assignment referred to in the *Tips and Tricks* section. He found the concept ambiguous at first. 'Next? Next what?'. Nevertheless, he understood the meaning once he started reflecting about what it could mean. It refers to what assignment will follow the current one.

### 8.4.2 The game creator

The first part, creating a game, was unproblematic. When he started to add assignments, though, the test person became confused. He assumed that he should follow the instructions, without realizing that they were not aimed at him. Instead, they are instructions to the player of the game, explaining what he should do in the first assignment.

Because of this misunderstanding, the user was looking for a notebook, and was confused when he couldn't find one. A prompting from me, where I explained that this was the first assignment of the game, set him on the right track.

When he reached the part where he had to choose a handler, he was confused again. It was evident that he had not grasped the handler concept clearly, which he confirmed upon questioning. He re-read the handler part of the manual, but was still unclear on the subject.

I offered the explanation that 'handlers determine what assignment comes next'. This was the information he needed to understand what handlers are. Still, he was unsure of whether he needed to choose a handler. 'Do I need a handler?', he asked.

He wanted to use the *custom* handler instead of the *minimal* handler which would have been the correct choice. His reasoning was that the user had to use a notebook, and that this was not specified anywhere. This is part of the next screen image, though, as he later discovered.
Choosing the optimal handler proved the most problematic task throughout the test. Whenever he was unsure of what to do, he would choose the minimal handler. His reason for this was that it was the default choice. He inquired after a default option like 'choose a handler', which you should be unable to select, but that would make it clear that you have to make a choice.

In the assignment where he ought to use a custom handler, the participant wanted to use a minimal handler. This was because there was no next assignment. I explained that this was to limit the number of assignments in the test game, and encourage him to imagine that the game would continue. When doing this, he correctly identified the custom handler as the right choice.

The user was also unsure about the difference between the handlers choice and number compare. He correctly used choice when it was appropriate, but in the assignment that required number compare, he was less sure. He initially wanted to use choice 'because you have to choose something' (the cost estimate for the project). In the end he decided upon number compare, because it involved numbers.

After having completed a few assignments, though, he became better at choosing the correct handler type. It seemed that the more he used the handlers, the better he understood the concept, and the easier it was to identify the best handler for each assignment.

Apart from these uncertainties regarding the handlers, most of the process of adding assignments went relatively smoothly. The user was unsure when making the first assignment, as he could not specify the next assignment, since there was no assignments to choose from yet.

At this point, he was unsure whether to press continue, or quit. He wondered whether the assignment would be saved if he pressed quit. In the end he chose continue, as is the appropriate choice.

He also conveyed that it was unclear what values were legal and what were not. This was particularly in relation to the time, money, and score values. In addition, he
wondered what the unit of *time* and *money* was.

As already mentioned, it became gradually easier for the test user to make assignments. Most of his questions and 'errors' came at the start of the test. The more assignments he had made, the more sure he seemed, and the less time it took to create a new assignment.

### 8.4.3 The game

When the test user played the game, everything went seamlessly. There was no questions or comments from the user. He experienced no difficulties, and there was no issues worth mentioning in this part of the test.

### 8.4.4 The interview

When asked about what he found missing in the software, he pointed out the need for a 'foolproof' user interface. He remarked that the average user will most likely make many mistakes, and that the software should prevent them from being able to make mistakes.

He also suggested more help texts, to make the software easier to use. Another recommendation was to add a *back*-button that could be used if it is necessary to go back and correct mistakes.

When asked about the need for a more graphical user interface, he said that the current interface was sufficient for the test game. However, he did acknowledge that it might be better with a more advanced interface if the game was larger. He also suggested that the appearance of the webpage be improved.

The participant confirmed my impression that it did take him a while to grasp the concept of handlers. He proposed to add an example of how to use handlers in the user manual to make the idea easier to understand. He thought that the names of the handlers were appropriate, and that the handlers gave him the flexibility needed to make a game.
CHAPTER 8. TESTING

To avoid the problem with referring to assignments that are not made yet, he suggested dividing the creation into two parts. First, you can add the basic information on all the assignments. Then, you can connect them through handlers.

The answer was yes when questioned whether he thought the units time, money, score were appropriate. He noted that similar values are used in different commercial games, and that they felt familiar. Although his own scores had not changed much during the game, he did feel that he was always aware of how he was doing because of these values.

When asked how the avatar picture influenced the game, the test user was positive. He pointed out that such use of pictures help make the game more interesting and fun.

8.5 Conclusions

From the test, it seems obvious that some of the concepts need a more comprehensive explanation. This is particularly true for the handlers. At the same time, it is clear that the test person did not study the user manual thoroughly.

It is difficult to inform the users about the structure of the game if they do not study the user manual. When I made the manual, I attempted to make it as complete as possible, to avoid misunderstandings and ease the game creation process. However, it might have resulted in an information overload, that there was too much text for the user to read.

It might be beneficial to divide the user manual in two. The first part should be a general one, where the structure of the game is explained. This is the part the user needs to study before creating a game.

The other part should be more of a reference, which the user can turn to if in doubt. Here, all the steps of the game creation will be explained in detail. For most users, this information is superfluous. However, some novice users may prefer to have such a step-by-step guide.
CHAPTER 8. TESTING

The current user manual has a similar structure, but the division between the parts is less clear. Also, as mentioned earlier, the first part needs to explain the game concepts more thoroughly.

In addition, there seems to be a need to define what values are legal in the web interface. It is apparently not sufficient to give this information in the user manual. Some of the explanatory text from the user manual should be duplicated on the web pages as well, to make sure everyone reads it.

From the test I conclude that the game generator has a somewhat high threshold to understand and use. It does need some explaining. However, after having completed a few assignments, the software is virtually self-explanatory. Although it takes some time to learn to use the program, it is easy to use once this learning period is finished.

The game itself seems easy to use. The user was given no instructions, and as mentioned earlier, he finished without any problems or questions. Consequently, I conclude that the game is sufficiently easy to use.

8.6 Summary

In this chapter I have conducted a think aloud evaluation of the system. Think aloud is a qualitative technique that illuminates what parts of the software is not intuitively understood or used.

The test revealed that some of the concepts were vague and needed more explanation. It also takes a little time to understand the software. Once you have understood it, though, the system is relatively easy to use.
Chapter 9

Discussion

In this chapter, I discuss whether the software I have developed meets the goals I specified in chapter 6 and the recommendations from chapter 3. I start by looking at whether my specifications are met. Then, I examine if the games that are created, have the most important features to make them appealing.

Next, I take a self-critical look at what could have been done differently during the course of the thesis. The chapter ends with the conclusions I draw based upon the previous discussion.

9.1 Does the software meet the specifications?

9.1.1 System design characteristics

User friendly
As the think aloud evaluation showed, it did take some time to get used to the concepts of the game and use them correctly. However, once the user has understood the basic components, there were few problems in using the game. Thus, what takes time to learn is the structure of the game, not the game generator software itself.

As discussed earlier, it would be unrealistic to expect to be able to build a game generator that does not require some basic training to use. If that were to be the goal, it would drastically reduce the complexity of the resulting games. The more
features that are included in the game, the more time must be spent studying the
game structure to understand it.

The game creation process went smoothly once the user had grasped the game con­cepts. This means that the game generator is relatively easy to use if you understand
the components of the game. The think aloud evaluation did point to some areas that
could be improved, though.

The main drawbacks the test user experienced with the user interface, were a cer­tain lack of explanatory text and pages that were not 'fool-proof'. This means that it
is the user's responsibility to ensure that he uses the system correctly. That makes
the software harder to use.

However, after the think aloud evaluation, I have improved the software. Now, all
fields that require a specific input contain information on what values are legal. All
fields required to fill in are also marked correspondingly. Figures 9.1 and 9.2 show
screenshots taken before and after the improvement, respectively.

Figure 9.1: Screenshot prior to improvement
While it is still possible to make mistakes, this does reduce the risk considerably. If in doubt, the user does not have to consult the user manual. Instead, all information he needs is on the webpages themselves. This makes it considerably easier to use the system.

A feature that was present even in the first version of the system, was the drop-down menus when choosing what assignments come next. They ensure that the game author cannot specify an illegal assignment as the next assignment.

In addition, this drop-down menu makes it easier to select the correct assignment. If the user were to input the assignment number, he could easily make a mistake. Now, in addition to assignment number, the name of the assignment is also apparent. This makes it even easier to identify the correct next assignment.

One way to further improve the user friendliness, would have been to have a graphical user interface. A 'map' over all the assignments, and where they lead might help the user envision the game structure. This may also make it harder to commit errors such as connecting the wrong assignments.
CHAPTER 9. DISCUSSION

While having a graphical user interface might have improved upon the user friendliness of the system, it is beyond the scope of this thesis. I have discussed the use of a graphical user interface further in section 10.1.1.

Even though there is little graphics, the software is sufficiently user friendly. The test shows that the system is relatively easy to use. It helps the user avoid mistakes, while providing him with an easy interface to make a game.

The test also showed that the game portion of the software is self-explanatory. As previously mentioned, the player received no instructions, yet he completed the game without any questions or problems. Thus, the goal of using the game without any training, is met.

The test person was familiar with playing games, though. For a novice player, the software might not be as easy to use. However, because computer games are so popular, the majority of informatics students will have some experience with playing. Consequently, the game is easy to use for the major part of the user group.

**Extendable**
The main feature that makes the software extendable, is the handler concept. Because the part that manages the player's answers is separate from the rest of the program, it is easy to change or add to this isolated segment.

This means that by understanding just a small portion of the program, you are able to add to the behavior of the game. The advantage of this, is that mainly one part of the program needs to be changed.

The structure of the game will need much less alterations, as the handlers are the driving force of the game. They are what takes the game from being a mere story you read, to being an interactive experience.

Another powerful component worth mentioning, is the custom handler. As previously mentioned, it allows the game author to add his own handler without altering
the code itself. This is important, as it allows you to drastically change what the game can accomplish without prior knowledge of the program.

Although this is a strength with the software, it can also potentially become a problem. Given an inadequate number of premade handlers, a lot of work will shift from the programmer to the game author. This would not be optimal, as it should be possible to create the game without programming knowledge.

Knowledge of programming should not be required of the author. And it would be too much to expect that he should know Python programming, or have to learn it, although it is a very intuitive language.

However, the last point, at least, can be helped. Because Python scripts can be combined with a wide range of other languages, it is possible to allow the handlers to be in a different programming language than Python. Although this is not a feature in the current prototype, it is a feasible extension for future versions.

To ensure that the game author does not end up with the responsibility for developing the handlers, it is important to put a lot of effort into making new handlers. Nevertheless, before this can be done, there needs to be some system development scenarios available. They will give a hint as to what type of handlers are needed.

If over-used, the custom handler can put too much responsibility on the game author. Still, the handler concept is still a great advantage. The alternative would be to incorporate this logic into the rest of the code. This would have made the process of extending the program much more tedious.

Another feature that makes the system easy to extend, is the design of the software. Both the game creator and the game itself uses the same data structures to store information. This makes it easier to extend the system, as you do not have to make changes to two different data structures. All changes to the main components will affect both parts of the system.
CHAPTER 9. DISCUSSION

Reusable
The main component in making the software reusable, is a data-driven, as opposed to a code-driven, system. This means that the game scenarios are completely separated from the code. Consequently, the game generator may be used to make more than one game.

The handlers also play a vital part in reusability of the software. Because you can choose how to manage the player's input, it is also possible to make different types of games.

If the needed handler type for a specific assignment is not among the premade ones, you can make your own using the custom handler. This makes the system very flexible, and allows the user to make games that are not related to system development.

Having more handlers, might have made the software more reusable. The more handlers, the easier it is to make different games. However, this will improve as more game scenarios are made. Each new game is likely to add new handler types to the system, as new needs originate.

The way the game content and the player's responses are stored, also help make the system reusable. Because the data is stored in XML, which is a very reusable language, it can easily be applied for other purposes. For instance, the game content can be integrated in a lecture to exemplify the system development process. Other uses are discussed in section 10

9.1.2 Functional requirements

The game generator
The intention
The game author can make different games using the same software. By changing the input, the games he creates differ from each other. Also, all the logic in the game is stored in data-files. Hence, the requirement that the system should be data-driven, is also met.
Type of game
The small test game used in the think aloud evaluation was a sketch of a system development game. The results from the test showed that the software does indeed allow the game author to create such a game.

User interface
The interface is web based, as specified.

Changing the game
The game author uses the same tool to both create a game and make additions to it. Also, altering a game is effortless. When changing an assignment, the old values are set as default input. Hence, the game author needs to alter only those parts of the assignments that he wants to change, without having to re-input the entire assignment.

The game

Type of game
The game consists of different challenges - assignments. They are the main building blocks of the game. In order to allow the player's responses to an assignment to influence how the game unfolds, I developed handlers. They determine what assignment comes next based on the player's input. Thus, he is partly in control of how the path through the game evolves.

User input
The game is able to handle different input types such as text, numbers and files.

Recording the player’s answers
To ensure that all the player's moves are recorded, the game saves this information in xml files. These files are easily read by humans, and contain all relevant information regarding how the player played the game. By using an XSLT stylesheet, you can transform the xml files into a format that is even easier to read, such as HTML. This transition is discussed in section 5.
Resumable
As the player should be able to quit the game at any time, the system writes its current state to file each time a new assignment is loaded. This guarantees that, if the player pauses the game, he can resume where he stopped. As an added benefit, this is useful in case of a computer crash. In such a case, minimal data will be lost, because the game state is saved so often.

Scoring
In order for the player to know how he is doing at all time, the game contains three different 'barometers' - time, money, and score. These values are always visible to the player, to ensure that he sees the consequences of his actions.

9.2 Is the game appealing for the players?

9.2.1 Challenges
To create the right amount of challenges, is mainly the game author's responsibility. However, the logic behind the software may help or obstruct this development.

The game generator allows the game author to adapt the game to the player's needs. The handlers allow the author to specify what assignment is next based upon the player's answers. This can be used to make the game easier if the player is doing poorly, or vice versa.

However, at present there is no opportunity to specify that the game should shift to an easier mode if the player’s score drops below a certain point. The score is the yardstick that indicates how well the player is doing. Hence, such a feature could be advantageous to help adapt the challenge to the player's own level.

This would be similar to first-person shooter games, where you at start-up usually may choose between several different levels of expertise. In some games, such as Unreal Tournament, you can specify that this level adjusts automatically, depending on how well you play.
On the other hand, for such a feature to be useful, the game author would have to make several different versions of the game. This would require a lot of extra time and effort, and might not be something the average game author is willing to do.

In addition, it is questionable whether such an approach is superior to the current solution. By taking advantage of the handlers, you are able to make the game easier or harder in the specific areas where the player needs it most.

If a player has trouble understanding UML diagrams, for instance, it might benefit him to get an easier assignment related to this type of diagram. This does not mean, however, that he needs the remainder of the game to be easier. The handler approach allows the game authors to adapt the game to the player's skills in the areas he needs it the most.

This is not possible if the score is the determinant of where the game is headed. With this technique, you can only change the level of the game if the player has done a consistently good or bad job.

This does have the advantage of catching those that may not have evident problems in specific areas, but rather are weak performers all-over. It might complement the handler's advantages, making an even more adaptable game.

However, being able to spot specific problem areas is more useful in an educational game. It can be used as illustrated above, by giving additional exercises in the fields where the student is weaker. Because of this, I have concluded that the handler solution is sufficient for my purpose.

Of course, this approach is rather simple. It allows the course of the game to change based on the player's answer to the previous assignment. It cannot do such advanced work as for instance making a profile of the students weak and strong points, and adapt the game to this.

While this, of course, would have been optimal, it also requires a significant portion of artificial intelligence. To develop such a logic is beyond the scope of this thesis.
Also, the current system can create challenging games that adapts to the player's skill level. Hence, it is sufficiently advanced to make an educational system development game.

9.2.2 Socialize

Because the game generator is designed to make single player games, this is not a very important point to consider. It might apply in the future if the system is extended to include multi-player games. I have discussed this point in detail in section 10.

9.2.3 Dynamic Solitary Experience

In order for the game to provide a *dynamic* experience, it must be possible for the player to influence the outcome. This is facilitated through the use of handlers.

The handlers determine what assignment comes next based on the player's answers to the current assignment. Thus, they allow the player to influence the game. Without the handlers, the game would be predetermined, rather than a dynamic experience. A dynamic experience is also related to interactivity, which is discussed in more detail in section 9.2.8.

9.2.4 Bragging rights

For the game to give bragging rights, it requires something that can set the different players apart from each other. Partly to support this, the game has three factors - *time*, *money* and *score*.

All three can be used to brag about your own achievements. This allows one person to claim the 'title' as the fastest player, while another had the highest score, for instance.

Another possible 'brag factor' could have been the time it takes to complete an assignment. If the game measured how much actual time the players spent on their tasks, this could be an additional element on which to compare scores.
This can be seen in first-person shooter games, such as *Doom*, where the game logs how much time you spend on each level. Adding such a feature to the system development game, however, can give a misleading impression.

In first-person shooter games, it is easy to log the time the player spends, as he cannot complete parts of the game without actually playing it. In a system development game, however, this is not the case.

It is quite probable that the players may pause the game in order to solve a complex task such as creating a UML diagram. Thus, there is no accurate way to determine the exact amount of time the player has spent on an assignment. Hence, this is not a good scale for comparing deeds, or to analyze how much time a player needs on a specific task.

However, it is possible to give the players even more opportunity to brag and compare achievements. By developing a webpage dedicated to the game, you can make it easier for the players to brag. This is discussed further in chapter 10.

But this has not been a focal point of my thesis - partly because there is no system development game scenario to make a webpage for, and partly because this task is completely separate from developing the game generator. It might be worth looking into in the future, though, as discussed in section 10.

### 9.2.5 Emotional experience

The responsibility of creating an emotional experience lays mainly with the game authors. It is the game story that determines to what degree the player’s emotions run high. However, whether the player is attached to his character does contribute to the emotional experience. How this is achieved, is discussed in section 9.2.11.

### 9.2.6 Explore

The current game generator does little to let the player explore a virtual world. This is mainly because the game is assignment-oriented rather than gameworld-oriented. The goal is not to explore a fictional game world, but to complete the given tasks.
Because the game is meant to be set in the real world, it is of little importance to provide a game world for the user. He already knows the world, what it looks like, and its rules. Nevertheless, it might have been advantageous to provide some sort of simple world where the player can move around.

At this point, however, this is less consequential. After all, the main feature of the game is its educational character. Therefore, focus has been on creating a system that can handle user input. Creating a graphical environment might be a task for the future, as mentioned in section 10.

Exploration can also be thought of as not only discovering new places, but new ideas and experiences as well. The software does take into account this part of exploring. It allows the player to delve into unknown territory such as meeting with clients, signing contracts and other aspects of the system development process.

### 9.2.7 Fantasize

It is the game authors' responsibility to ensure that the player's desire to fantasize is satisfied. This is an area where the technical aspects play but a minimal part. Yet, how easy it is to fantasize in a game is in part a matter of the ability to identify with the main character. As discussed in section 9.2.11, the software does contribute to this part of fantasizing.

### 9.2.8 Interactivity

Whether the game is interactive or not, depends on the game author. You can make a game that gives no freedom whatsoever, and that railroads the player from start to finish. However, the game generator also allows for a game where the player can influence the storyline.

The handlers allows for this interactivity. As discussed thoroughly above, this concept makes it possible for the game to adapt to the players. Thus, the player's choices will determine what assignment comes next.
Although the players can influence the way the game unfolds, this might not be evident to them. From the players' perspective, they are faced with assignment after assignment. They cannot choose between different assignments, and they are not aware that their actions do in fact determine what assignment comes next.

This may seem like railroading to the players. A solution might have been to allow the players to choose between different tasks. Although this would give the players more freedom, it is not a good simulation of the system development process.

When developing systems, the different assignments are usually done in a certain order. It would therefore not be realistic to give the players too much freedom to choose their own tasks. After all, the goal of the game is to make a simulation of a system development process that is close to real life.

Yet, some of the steps in the development process might require the people involved to choose between several different tasks. This can be simulated by using the Choice handler, where the player chooses between different options. Hence, it is possible to give the players more freedom if so desired.

As we see, there is a difference between perceived interactivity and actual interactivity. The game does in fact have a high degree of interactivity, even though it is not always apparent to the player.

Some elements help make the game feel more interactive to the player. The player is always aware of how well he is doing in relation to time, money and score. These three values will rise or fall depending on his moves. This helps support the impression that the player's decisions make a difference to the storyline.

9.2.9 Getting better at something

This point is primarily the responsibility of the game author. The quality of the challenges he gives the players will determine whether they have the chance to improve or not. Yet, the software does help the author with this work.
As noted earlier, the *handler* concept allows the game author to adapt the game to the player's skills. Creating challenges that fit his level, makes it more likely that the player will improve. Thus, the game structure builds the foundation for making challenges that allow the player to get better at one or more subjects.

Also, the player is always aware of how he is doing with regard to the scales *time*, *money*, and *score*. This can help him feel that he is improving, as he has a visual confirmation stating how well he is doing. If nothing else, he can improve by heightening his score.

**9.2.10 Clear and compelling goals**

Again, this is connected to the storyline rather than to the technical platform. Whether the goal is compelling enough for the player, is determined by the story of the game and not by any technical aspects. However, if the goal is tied up to the main character of the game, it is easier for the player to embrace the goal as his own if he identifies with the character. The software does help with this identification, as discussed in section 9.2.11.

**9.2.11 Identification with the main character**

The responsibility to ensure that the player identifies with his character lies mainly with the game author. Still, some technical aspects can make it more likely that the player will empathize with the character he plays.

The character in the game is represented by an image. When starting a new game, the player can decide what face he wants to personify himself. In the prototype, he can choose between male and female character, as well as different skin-tones and hair-colors. Figure 9.3 shows the avatars available.

The idea is that by allowing the player to choose between different faces, he can pick the one he prefers. This will increase the chance that he identifies with the character.
9.2.12 Providing feedback

The main method by which the game provides feedback to the player, is through the three yardsticks - *time*, *money*, and *score*. The levels will rise or fall depending on the quality of the player's answers.
After an assignment is completed, the three scores are adjusted according to the player's solution to the task. This makes it easy for the player to follow his progress, and always know whether he did well or not.

Of the three scales, *score* gives the player the most accurate indications of how he is doing. *Time* and *money* can mislead a little as they will usually drop whether the player has done well or not. How much they change is not necessarily a reflection of the quality of the player's choices. For instance, hiring more employees will be expensive, but can be a wise decision nonetheless.

Although *time* and *money* are not always good indicators of whether the player has made wise decisions, they are still important parts of a game. When developing a system, there are always constraints on the time and money available.

### 9.3 What could have been done differently?

As explained earlier, I have worked on the technological platform of the system development game, while the task of creating scenarios is the theme of another master's thesis. As a consequence, I have had no finished game scenario on which to build the game.

I would have preferred having a complete game scenario to study. Although the structure of the system would most likely be quite similar to what it is today, it would have made it easier to make more handlers geared specifically to a system development game.

I was given a short, preliminary game scenario by my supervisor, to get an idea of what a system development game might contain. This helped me create the structure of the game. But, being relatively short, it revealed only a few handler types.

Ideally, a complete system development game scenario should have been created before attempting to develop a technological platform for the game. This would have shown more of the handler types that are needed to create such a game. The technical platform might have been more complete with this information.
CHAPTER 9. DISCUSSION

Yet, having a technical platform to refer to, could make it easier for the master student who seeks to create the game scenario. Both parts are essential in creating a good game. Since it is easy to make new handler types, the lack of a game scenario should not yield any critical consequences.

It would also have been advantageous to perform a few more tests. This might have given even more insight into how the user views the system. In addition, it could have uncovered more areas that needed improvement.

Unfortunately, qualitative tests are time-consuming. Because of the limited time available, I had to concentrate on other parts of the system. But, it is my belief that the test did succeed in uncovering the major weaknesses of the system.

9.4 Conclusion

As is evident from the preceding discussion, most of the specifications from chapter 6 and 4 are met. Likewise, the software helps bring forth most of the appealing qualities of a game. Yet, there is some room for improvement.

The decision not to put any focus on graphics has, of course, had some impact on the user friendliness. The game generator would have been easier to use with a more graphical user interface, as discussed in chapter 10. Despite some imperfections, in the main the software is easy to use.

The concept of handlers is vital to several of the aspects discussed above. This is evident from the fact that the handler plays a crucial role in many of the points examined in this chapter. It lies at the core of the structure, and is the single most important part of the system. Without this feature, the usability of the software would be drastically reduced.

As detailed earlier in this chapter, also other characteristics of the system make it useful. Although some areas can be further improved, the software mostly meets the criteria set earlier. Hence, the main goals for this system are achieved.
9.5 Summary

In this chapter I have discussed whether I have achieved my goals for the software. The software mostly meets the specifications from the design phase of the development. It also supports the game author in making an interesting and fun game.

A few things could have been done differently. But these are minor points, as my main goals were achieved. The software meets all my requirements, and it allows the author to make appealing games.
Chapter 10

Future work

In this chapter, I look at the future of this software. First, I discuss what are the most important fields to focus on to further develop the program. Then I suggest some new areas of use for the software.

10.1 Important improvements

The game generator that I have developed, is meant to be merely a prototype, not a finished software. Consequently, the areas that will be discussed in this chapter, were never intended to be a part of the prototype. Yet, having spent much time making the software, I have some reflections on what it takes to develop a complete product from the prototype.

10.1.1 GUI

The perhaps most important feature to focus on in the future, would be to create a true graphical user interface (GUI) for the authoring tool. The current version is considerably more user friendly than a command line interface, or having to manually edit the game files. However, by adding a GUI, it will be significantly easier to create a game. This will make the tool more appealing to the users.

While it would have been both interesting and useful to create a real GUI for the application, it is beyond the scope of this thesis. This is a task that requires extensive
research of its own. Nevertheless, because this is such a vital aspect, I have made an attempt to outline the most important features that need to be included.

Two major attributes are vital to a game creator GUI. It is desirable to have a graphical presentation of the game, showing all the assignments, and the possible paths the player can take. Another vital point, is that it needs to be more intuitive to change assignments.

**Graphical representation of the game structure**

A graphical representation of the structure of the game, will make it easier for the authors to picture the different paths of the game. This can prevent them from making logical errors, such as for example creating a never-ending loop the players can be trapped in.

Such a loop will occur if, for instance, assignment 1 leads to assignments 2 which leads back to assignment 1 again. This type of error is easily overlooked with the current interface. However, a graphical representation would eliminate the problem, as it will be blatantly evident for the author if he ever makes a game with such a loop.

Another advantage with a graphical presentation of the game, is that you see all the different possible paths very clearly. This makes it apparent how many assignments there are in the longest or the shortest path through the game, for instance. This could, in turn, ease the task of creating better games and designing them exactly as one wants.

In figure 10.1, I have made a sketch of what such a graphical representation can look like. This is purely meant to present the important elements, not to suggest how the final presentation will appear.

It will take a lot of skill and effort to make a user friendly representation of the game, and I have not attempted to do this here. However, the figure does contain the key elements of such a representation.
Obviously, the most important components are the assignments, and the connections between them. The assignments should be represented with only the information necessary for the user to instantly be able to differentiate between them.

Examples of such information can be assignment id and name. However, just what information should be included have to be researched thoroughly. Too much can result in information overflow, and too little can lead to confusion for the user. These are some of the issues that need to be addressed when designing a GUI.

The connection between the assignments should also be presented graphically. Here, it would be natural to draw an arrow from one assignment to the next. It should be evident which direction the player can move in. Consequently, a line will not suffice, as it does not determine the direction.

It might also be desirable to add information to this connection. For instance, it might be valuable to indicate which conditions will lead the player to that particular assignment.
As an example, answering 'yes' to an assignment might lead to assignment A, while answering 'no' leads to assignment B. Again, it must be considered whether this makes it easier to use the tool, or whether it will confuse the user.

**Changing assignments**

This point is closely related to the previous one. With a graphical representation of the game structure, it will already be easier for the user to change assignments. Having a quick overview of the game available makes it more apparent what needs to be changed.

However, there are some measures that can be taken to ease this task even further. One idea is to allow the user to choose what assignment to change by clicking on the graphical representation of the assignment in question.

This should in turn result in a sort of pop-up window with fields where the user can fill in information on the assignment. This part can be similar to the existing change assignment feature of the current version of the game creator. It should also be equally easy to make a new assignment.

The main problem with creating a new assignment in the current version, is that it is not initially connected to any other assignments. In effect, this means that it is not a part of the game until it is linked to an existing assignment. To create this connection, you need to add it to the already existing assignment.

With a GUI, it should be possible to create this link without having to change the whole assignment. Optimally, it should be feasible to create a connection simply by clicking on the two assignments. This could result in a pop-up, that allows you to change the path by filling out a few fields.

Such a feature would improve the game creation greatly. In addition to reducing the risk of errors, this will make the program seem more appealing to the authors.

As I have discussed earlier, there are a lot of issues to consider when making a GUI. The limited time available for this thesis made such work impossible. However, the...
elements I have mentioned above should be at the core when designing a possible future graphical user interface for the game creator.

10.1.2 Handlers

Another area worth focusing on in the future, is the **handlers**. They are at the core of the software, and the main driving force of the game. With the right handlers, there are virtually no limits as to what user input the game is able to manage.

However, at present, the choice of handlers is limited. They are meant to show the possibilities of the software, rather than representing the complete set of options available in the final edition of the software.

At present, there are no complete system development game scenarios available on which to test the handlers on. When such a storyline is finished, it will be easier to see what types of handlers are needed. Naturally, the types will differ from game to game.

One important task will be to look at different system development scenarios to determine what types of handlers are generic enough to be applicable to most games. The next step will then of course be to actually develop these handlers.

A type of handlers that might be advantageous to develop early on, is tools that evaluate system development documents. One example can be a small program that compares a UML diagram to the correct version, and determines the quality of the answer based on the difference between the two.

Because the current software is merely a prototype, it would also be beneficial to improve some of the existing handlers. The **Question handler**, for instance, is very simple. Determining what question the player poses, even though he might not use the exact wording of the answers-file, can be a challenging task.

Improving current handlers and adding new ones, is probably the one change that will have the most noticeable effect on the game. After all, the handlers determine
what type of input the game can manage. By strengthening this part, the flexibility of
the game is increased.

10.1.3 Graphics and sound

In commercial games, one of the most prominent features is often the graphics. Stun-
nning worlds, advanced 3D graphics, and convincing sound effects are something the
players have become accustomed to. Although the system development game envi-
ronment is not completely text-based, there are few graphical elements.

By adding more pictures and animations to the game, it will become more interest-
ing to the players. It seems too ambitious a goal to hope that students will prefer
an educational system development game to a commercial counterpart. However, by
copying some of these games’ focus on graphics and sounds, the simple game can
become more appealing.

A few small changes may help with the overall impression. This can be adding
sound-effects such as cheering when the player has made a wise move, or more illus-
trations. In addition, there are some more extensive ideas worth looking into.

Talking heads

Talking heads (Talking Head Demo n.d.) is an interactive speech animation product
made by the Norwegian company Inovani (Inovani n.d.). This is a software where
animated characters are able to 'talk'. A picture taken from a demonstration on their
webpage can be seen in figure 10.2

As is apparent in the figure, there are several different characters to choose from.
Although there are cartoon versions as well, the more life-like human heads would
probably be most appropriate for a system development game.

In addition, you are able to choose what voice the character will speak with. There
are both male and female, adult and child versions available. You can also select the
language.
After having decided on the character, you determine what it will say. You are even able to indicate that the character will use facial expressions to display moods such as for instance happy, sad or surprised. The character will then say the text you have specified, with corresponding facial movement.
These talking heads can be used when the player is in meetings, for instance. Each time the player's character interacts with a person, he or she can be 'played' by a talking head. It is also possible to let the main character be represented by one of the characters. It can then be evident how well the player is doing by the facial expression of the talking head, for instance.

This technology can help bring life to a system development game. Using animated characters to express a message, rather than letting the player read a text, will make the game more interesting to play, as well as add realism.

**A virtual world**
One possibility is to create a virtual world in which the player can move around. At present, the character is presented with tasks that must be completed. There are few references to the game world, as the adventure is assumed to be set in our own world.

To add more realism, and make the game more appealing, it could be worth the effort to create a graphical representation of the surroundings. Thus, it will be easier for the player to be immersed in the game.

However, as a system development game is focused more on completing tasks, rather than exploring a fictional world, this point should not be the main focus. Such a virtual world should be a background that makes the game interesting, rather than being the main focus.

**Efficiency**
If some of the suggestions above are set into action, it might be necessary to optimize the code. Because there are so few graphical elements in the game at present, there has been no need to look too closely at the efficiency of the code.

However, as more graphics are added, the need for efficiency increases. Consequently, it might be an issue worth taking into consideration in the future.
10.1.4 Making the game a more social experience

A multi-player game
At present, the game is single-player. However, it might be interesting to research whether it has a potential as a multi-player game as well. This can enhance the appeal of the game, as playing against others adds a social element to gaming.

There are several ways in which the game might become multi-player. One option is that the players’ characters work in competing firms. They all have to fight to get the same client, and the one who does the best will win.

Another alternative is to let the players be employees within the same firm, but with different responsibilities. In this case, the players might cooperate to get the job done. This can have the additional advantage of teaching group dynamics and interacting with others in the workplace.

An interactive webpage for the game
While making a webpage for the game is not directly related to developing the game, it might actually improve the game experience for the players. Most commercial first person shooter games have webpages dedicated to the game. Here, the players can discuss the game, brag about how well they did, and help each other out.

The popularity of such game-related sites shows that this is an important part of the experience of the game. It would also help enhancing the social experience, as well as let the players brag about their accomplishments.

Such a webpage might contain discussion forums, where the players can interact. In addition, there can be scoreboards showing the top 10 players, for instance. This can help to increase the competitiveness among the players, and thereby make the game more appealing.

10.1.5 Developing a storyline for the game
Naturally, without a storyline, all the suggestions above are useless. After all, the storyline is the core element, without which there is no game.
At present there is no finished storyline available for a system development game. The most important step will of course be to develop one. This will also ease the expansion of the technological platform, as it will become more evident what improvements are necessary.

Because the storyline is not a part of the technological platform, I will not go further into this point here. It suffices to mention that it would be beneficial to develop one or more larger scenarios as soon as possible.

10.2 Other uses

10.2.1 Reusing the game content

Both the game content and the players' answers are stored in a manner that makes it easy to reuse them in a different context. For educational purposes, this can be very useful.

One possible way to make use of the game content, can be in system development lectures. There, the lecturer can go through the different assignments, and explain what the best choices would be in this situation. The game can then become the focal point of a lecture, and giving the students examples of good and bad system development practices, and their consequences.

It is also possible to use the players' answers to the game in a similar way. Although the game does provide feedback on the quality of the players’ choices, it might be advantageous to learn more about how their answers affected the game.

Here, the lecturers (or teacher's assistants) can go through the answers with the student, and explain in detail what was positive and negative about his solutions. This might heighten the student's understanding of the system development process.

These are just a few examples of how the game can be reused in different ways. Because the file structure is developed with the goal of being reusable, only the ima-
CHAPTER 10. FUTURE WORK

Imagination sets limits to different uses. The important part, is that this technology allows the game to be more than just a game.

10.2.2 Different types of games

The game generator is built to be a tool to create system development games. But as the software is built to be reusable, it is not limited to making this type of game.

Granted, this software will never be used to develop games that require vast amounts of graphics, such as first person shooter games. Neither is it suited to produce rule-based games such as Chess or Tetris. However, there are some other types of games that can be made with this tool.

A genre that the game generator might be suited for, is knowledge games. Here, it can be used to make more advanced and dynamic games, where the next question or task is determined by how well the player did previously.

Another type that can be made from this tool, is simple adventure games. This is the type of text-based game where you are presented with a quest, and you can choose how to solve the different steps of it. It is similar to a system development game, but the tasks are not educational, but 'just for fun'.

These are a few of the other types of games that can be created with the game generator. It can be used to create most types of games where the game world or the rules are not the most prominent features of the game.

10.3 Summary

Because the software is only a prototype at the present time, it is important to consider how to develop it further. Some important areas to explore, are developing a graphical user interface, making more handlers, adding more graphics and sounds, strengthening the social experience of the game as well as developing a storyline scenario for the game.
The work of this thesis can serve other purposes than just generating a system development game. The game content and the players’ actions can be used in education, to give a greater understanding of the system development process. In addition, it is also possible to use the software to generate different types of games.
Chapter 11

Conclusion

Games play an important part in today’s society. By taking advantage of this, and creating an educational game, you can make the system development process more interesting and understandable to the students.

In this thesis, I have considered the technical aspects of making a game that simulates the system development process. I have looked at how to design games, and what makes them exciting to the players. All these ideas have culminated in my own game generator software.

I have designed and implemented the structure for a system development game. In addition, I have developed a game generator that makes use of this game architecture. By feeding the software with the storyline for a game, you get a finished game that can be played online in a web browser.

My main goals for the software, were that it should be user friendly, extendable and reusable. In addition, it was important that the resulting game would be appealing to the players. This is mainly the responsibility of the game author, but to a certain degree, the technological platform can support this.

In my system, a game consists of many assignments. Each assignment is a task the user must solve. Because the player's actions should somewhat determine what happens in the game, I developed the concept of handlers. They determine what is the next assignment based on the player's answer to the current assignment.
The handler concept is at the core of the system. Because they are separate from the rest of the code, they are easily extended, added to, or changed. This makes the software easy to improve in the future, if the need for different handler types should arise.

By separating the assignment from the handler, you have a more flexible and reusable system. Now, each assignment is connected with the appropriate handler, chosen among a set of pre-made handlers.

This is far better than having to custom-make the handler for each assignment. In my approach, the logic of how the player’s input affects the course of the game, is kept separate from the assignments’ content.

I have developed several different types of handlers to demonstrate the functionality of the concept. The custom handler is worth taking special notice of. It is a handler that allows the game author to make his own handler if none of the default handlers are applicable. This is a great strength, as it lets the game author extend the system without changing it.

Another related point, is that the game is data-driven. This means that the logic resides in the data, rather than in the system. Consequently, by changing the data, you change the game. This feature, along with the handlers, is what makes the software a game generator and not merely another game engine.

The entire system is built upon the idea of separating logic from content. The assignments are separate from the handlers that determine the course of the game. The game storyline is separate from the code. This division is vital in order to make a game generator.

In addition to these essential architectural choices, the software is also relatively easy to use and learn for the user. The qualitative testing that I conducted, confirms that it is easy to make a system development game using the game generator. The resulting game is virtually self-explanatory to use.
The system meets all the requirements I have specified. I conclude that the task to develop a technical platform for a system development game has been successful. The strengths and flexibility of the software make it an excellent tool for creating an educational simulation game.
Bibliography

*http://dictionary.reference.com/browse/avatar


*http://www.dagbladet.no/kultur/2007/06/07/502887.html


*http://www.w3.org/

*http://www.w3.org/Consortium/


*http://www.adobe.com/products/dreamweaver/
*http://www.businessweek.com/technology/content/jun2005/tc20050623_2687_tc024.htm/

*http://www.ebay.com/

*http://www.eclipse.org/

*http://game-editor.com/

*http://dictionary.reference.com/browse/edutainment

*http://xs215.xs.to/xs215/07223/epic_mount.jpg

Games at work may be good for you (n.d.), BBC. Last visited July 2007.  
*http://news.bbc.co.uk/2/hi/technology/3247595.stm

*http://www.aftenposten.no/nyheter/uriks/article1729136.ece

*http://www.inovani.no/

*http://jonpy.sourceforge.net/

Juul, J. (2003), Looking for a heart of gameness, Utrecht University.

*http://kart.nexon.com/

*http://www.dagbladet.no/kultur/2006/12/03/484855.html

*http://www.lectora.no


*http://www.php.net/


*http://www.problemsandprogrammers.com/

*http://pydev.sourceforge.net/

*http://www.python.org/

*http://www.riventi.com/

*http://www.riventi.com/download/screenshots/LIPS.Application.jpg


  *http://www.ics.uci.edu/ emilyo/SimSE/


  *http://www.dagbladet.no/kultur/2007/01/31/490529.html

  *http://subversion.tigris.org/


  *http://www.facems.com/Examples/Personalities/index.php


  *http://www.w3schools.com/xml/

  *http://www.w3schools.com/dtd/

  *http://www.w3schools.com/xsl/
BIBLIOGRAPHY

*http://www.worldofwarcraft.com


*http://www.wow-gold-price-list.com/
Appendix A

Contents of the CD

The CD that accompanies the printed version of this thesis, has the following main structure:

A.1 thesis

This subdirectory contains an electronic copy of this thesis.

A.2 src

This subdirectory contains the source code for my system.

A.3 doc

This subdirectory contains the documentation for my system.

A.4 test

This subdirectory contains the audio recording of the think aloud evaluation
Appendix B

User manual

This user manual was the one used in the *think aloud* evaluation of the system. It was designed for this test, and is not meant to be a stand-alone user manual.

B.1 The structure of the game

B.1.1 Counters

There are three counters that indicate how well the player is doing. This is score, time and money. Score indicates how good the player's solutions to the problems have been. Time says how much virtual time the player has left to complete the game. Money expresses how much virtual money the player has left. These three will rise or fall during play, depending on the player's choices.

B.1.2 Assignments

The game is composed of assignments. When making a game, you first need to break the scenario into different assignments. An assignment is a specific task that the player must complete.

Some assignments may be purely informational, requiring no input from the player. In others, the player will have to solve a problem or give his opinions.
One assignment is followed by another until the end of the game. How the player solves one assignment, determines what assignment comes next. The player does not have to play all assignments, as one assignment might have several possible next assignments. However, the player will only end up with one of them, depending on his solutions.

### B.1.3 Random assignments

Random assignments are assignments that are not a part of the game’s storyline. Instead, they may happen at (almost) any time. They may help the player (such as give him more time), or hinder him (for instance his computer crashes).

### B.1.4 Handlers

The handlers are the part of the system that manages the player’s input. They are small subprograms that determine how the player’s decisions influence the game.

This is where you specify how each choice the player makes affects his score, the available time and money, and what assignment will follow.

Each assignment has its own handler. There are several different types of handlers, and which you use will depend on what you want to achieve with the assignment.

**Minimal handler**
This is the simplest handler. It is used when the next assignment is set regardless of what the player answers. This can for instance be in connection with pure information assignments.

**Choice handler**
With this handler, the player has to choose between several predefined options. This can for example be used if the player should determine which of several people he wants to talk to.

**Number compare**
Here, the player’s input is compared to a predetermined value to see if it is higher,
lower or equal to it. One use for this might be if the player should give an estimate of the cost of a project.

**Question handler**
This is used when the player interacts with other people and asks them questions. It allows the system to find the answers to the player's questions. This can be useful if the player is in a meeting, for example.

**Custom handler**
Here you can make your own handler. This can be used when none of the previous handler types are applicable.

### B.1.5 Game over

The game can end in one of four ways.

**Completing the game**
When there are no more assignments for the player to play, he has completed the game. This is the ending where the player 'beats' the game.

**Too low score**
If the player's score falls below a specified limit, the player looses the game because he has made too many bad decisions.

**Not enough money**
If the player runs out of money, he loses the game.

**Not enough time**
If the player is out of time, he loses the game.
B.2 Making a game

To make a new game, choose 'Make a new game'. Then fill in the following fields.

**Name**: Text
Input the name of the game here.

**Random level**: Percentage (0.0-1.0)
Indicates what the percentage chance is that the next assignment is a random assignment. This number determines how often a random assignment occurs.

**Score limit**: Percentage (0.0-1.0)
If the player's score drops below this point, the game will end. The player starts with a score of 0.5. Put -99 if you do not wish to use this feature.

**Start time**: Number
Determines how much virtual time (in hours) the player has to complete the game. Put -99 if you do not wish to use this feature.

**Start money**: Number
Determines how much money the player has available at the start of the game. Put -99 if you do not wish to use this feature.

**Administrators' email**: Email addresses
The email addresses of the game administrators. They will receive emails when the game is unsure of how to handle the user input.

B.3 Making an assignment

If you have just created a new game, you can skip right to the next paragraph. Otherwise, follow the instructions. To make an assignment, choose 'add to an existing game'. Next, choose the game you want to add assignments to from the drop-down menu. If you have just created a new game, you can skip the next paragraph.
Choose 'change existing' if you want to make changes to an assignment. Or choose 'make a new one' to make a new assignment.

B.3.1 Regular assignment

Part 1

Name: Text
The name of the assignment.

Type: Drop-down
The type of assignment. Choose between the allowed types from the drop-down menu.

Content: Text
The textual context of the assignment. This text will describe the task to the player.

File: File (optional)
Here you can upload any file you want to show to the player. This might for instance be a .pdf, a video, or a soundclip.

If you chose assignment type 'random', go to section B.3.2 for the next step. Otherwise, continue with the instructions below.

Part 2

Handler: Drop-down
Choose what handler type you want for this assignment from the drop-down menu.

Random: Yes/No
Indicates whether this assignment can be followed by a random assignment or not.
The next step depends on which handler you want for your assignment. Please refer to the appropriate handler in section B.3.3 for instructions on completing the final step.

### B.3.2 Random assignment

Random assignments are assignments that might happen at (almost) any time, and that are not part of the game’s storyline.

**Effect:** Drop-down  
Does this assignment help (good) or obstruct (bad) the player?

**Time:** Number  
How much time has this choice taken?

**Money:** Number  
How much money has been spent on this choice?

### B.3.3 Handlers

**Minimal handler**

**Input type:** Drop-down  
Determine what type of input, if any, the player may give in response to this assignment. Choose the appropriate type from the drop-down menu.

**Next:** Drop-down  
What assignment follows this one? Choose the next assignment from the drop-down menu.

**Time:** Number  
How much time has this assignment taken?
**Money**: Number
How much money has been spent on this assignment?

**Score**: Decimal number (1.0 - (-1.0))
How has the score of the player changed during this assignment? 0.0 is no change, a negative number means the player has done badly, while a positive number means the player has done well.

**Choice handler**
This handler specifies different choices the player is given in this assignment. He can choose one of the options, or choose the unspecified 'other'.

**Value**: Text
Each value represents a possible answer the player might give to this assignment. *Other* has no value, as it includes all other answers not specified.

**Next**: Drop-down
What assignment follows if the player chooses this option? Choose the next assignment from the drop-down menu.

**Time**: Number
How much time has this choice taken?

**Money**: Number
How much money has been spent on this choice?

**Score**: Decimal number (1.0 - (-1.0))
Based on his choice, how has the score of the player changed? 0.0 is no change, a negative number means the player has done badly, while a positive number means the player has done well.

**Number Compare**
This handler supplies a reference number, and you can specify what happens if the user chooses a number above, below or equal to this number.
**Number:** Number
Determines the reference number which the player's answer is compared to.

**Next:** Drop-down
What assignment follows if the player chooses this option? Choose the next assignment from the drop-down menu.

**Time:** Number
How much time has this choice taken?

**Money:** Number
How much money has been spent on this choice?

**Score:** Decimal number (1.0 - (-1.0))
How has the score of the player changed based on his choice? 0.0 is no change, a negative number means the player has done badly, while a positive number means the player has done well.

**Question handler** This handler allows the player to ask questions which will be answered by the system.

**Answers filename:** File
Upload the file where the answers to the possible questions are stored. The file should be in the following format.

```xml
<questions>
  <question>
    <keyword>question number 1 here</keyword>
    <answer>answer to question number 1 here</answer>
  </question>
  <question>
    <keyword>question number 2 here</keyword>
    <answer>answer to question number 2 here</answer>
  </question>
</questions>
```
Questions reference: Drop-down
In what assignment has the player taken notes of what questions he will ask? Choose the appropriate assignment from the drop-down menu.

Next: Drop-down
What assignment follows if the player chooses this option? Choose the next assignment from the drop-down menu.

Time: Number
How much time has this choice taken?

Money: Number
How much money has been spent on this choice?

Score: Decimal number (1.0 - (-1.0))
Based on his choice, How has the score of the player changed? 0.0 is no change, a negative number means the player has done badly, while a positive number means the player has done well.

Custom handler The custom handler allows you to specify your own file to manage the player’s input.

Handler file: File
Upload the file where your own handler is located. It must be on the following format:

```
#!/store/bin/python

def processCustom(input):
    next = None
    time = 0
    money = 0.0
```
badness = 0.0

# insert your own code here
# input is the player's answers, what type it is depends on
# what you have specified in the assignment

return next, time, money, badness

Next: Drop-down
What assignment follows if the player chooses this option? Choose the next assignment from the drop-down menu. Specify all the different next assignments that may follow.

B.4 Tips & tricks
Because you can only set 'next' to assignments that are already made, it is recommended that you first make assignment number 1, then start from the last assignment and work your way backwards. If you make assignments from start to finish, you must then go back and change the assignments, to make sure that they lead to the correct next assignment.
Appendix C

The test game

This rudimentary game was developed to test the game generator software. It is not intended to be a finished system development game.

C.1 A system development game

Name: A system development game

Random level: 0.5

Score limit: 0.0

Start time: 100

Start money: 100 000

Administrators' email: aclandro@ifi.uio.no
C.2 Regular Assignments

C.2.1 Assignment A

Name: A job offer

You are employed in a system development consultant firm. One day you find the following email in your inbox.

Your company wants the job, and you sit down to make notes on questions to ask at the meeting. Use the notepad below.

(Attach joboffer.pdf)

Can be followed by random assignment: No

Next: Assignment B

Time: 1
Money: 0
Score: 0.0

C.2.2 Assignment B

Name: The orientation meeting.

The leader of the Ruritania Artist Society explain that they want a system that keep track of all exhibitions, where they are, what artists are involved, and a full list of all the pictures that are being exhibited. They also want to be able to quickly find out what galleries are available for new exhibitions at a certain time. He then opens for questions, and this is the time to ask the questions you have noted on your notepad.

Can be followed by random assignment: No
C.2.3 Assignment C

Name: Proposing an estimate for the cost of the project

Now it is your job to give an estimate of how much your company would demand for this job. Your bid will determine whether your company gets the job or not.

Can be followed by random assignment: Yes

If the estimate is 100.000:
Next: Assignment E
Time: 5
Money: 0
Score: 0.0

If the estimate is less than 100.000:
Next: Assignment E
Time: 5
Money: 0
Score: 0.0

If the estimate is more than 100.000:
Next: Assignment D
Time: 5
APPENDIX C. THE TEST GAME

Money: 0
Score: -0.3

C.2.4 Assignment D

Name: Someone else got the job

Another company bid less than you did, and so they got the job. You have now lost a potential contract for your firm.

Can be followed by random assignment: No

Next: None
Time: 1
Money: 0
Score: -0.5

C.2.5 Assignment E

Name: You got the contract

Congratulations! Your company got the contract. You have to determine where to start the project. You can either meet with the Ruritania Artist Society, model the system, or take a vacation.

Can be followed by random assignment: Yes

Meet with the Ruritania Artist Society:
Next: Assignment F
Time: 1
Money: 0
Score: 0.3

Model the system:
Next: Assignment H
Time: 1
Money: 0
Score: 0.1

Take a vacation:
Next: Assignment I
Time: 1
Money: 0
Score: -0.3

Other:
Next: None
Time: 0
Money: 0
Score: 0.0

C.2.6 Assignment F

Name: Preparing for the meeting with the RAS

Before your meeting with the Ruritania Artist Society, you sit down and think about what you need to know to build the system. Mark your questions on the notebook below.

Can be followed by random assignment: No
Next: Assignment G
Time: 5
Money: 0
Score: 0.0

C.2.7 Assignment G

Name: Meeting with the RAS

You meet with members of the RAS, and they explain what functionality they want from the system. After having listened to them, you are free to ask the questions you noted earlier.

Can be followed by random assignment: No

Next: Assignment H
Time: 5
Money: 0
Score: 0.0

C.2.8 Assignment H

Name: Modelling the system

Your first job as project manager is to create a UML diagram of the system.

Can be followed by random assignment: No

Next: None
Time: -
Money: -
C.2.9 Assignment I

Name: Demoted

Because you decided to take a vacation at a crucial point in the project, you are demoted.

Can be followed by random assignment: No

Next: None
Time: -
Money: -
Score: -
C.3 Random assignments

C.3.1 Random assignment 1

Name: Sudden illness

You and most of your crew are hit by the flu. You all have to stay home for a week.

Effect: Bad
Time: -40
Money: 0
Appendix D

Test files

D.1 faq.xml

<questions>
<question>
<keyword>My question 1</keyword>
<answer>My answer 1</answer>
</question>
<question>
<keyword>My question 2</keyword>
<answer>My answer 2</answer>
</question>
</questions>
D.2 custom_handler.py

#!/store/bin/python

def processCustom(input):
    next = None
    time = 0
    money = 0.0
    score = 0.0

    return next, time, money, score
FROM: The Ruritania Artist Society (RAS)

TO: Several different system development companies

SUBJECT: An information system for our galleries

We, the Ruritania Artist Society would like to have an information system that will help us organize our galleries and exhibitions. We would like to invite your company to an orientational meeting regarding this opportunity.

Date: Tomorrow
Time: 13.00
Place: The Ruritania Artist Society building