xylos

Creating a game for the Imagine Cup 2009

Thesis Bachelor Project IN3405

P. W. G. Brussee 1308025
N. Kraayenbrink 1308149
S. P. van Sambeek 1308289

Delft University of Technology
Faculty Electrical Engineering, Mathematics and Computer Science

June 29, 2009

Exam committee:

Dr.ir. A.R. Bidarra
Ir. M. Sepers
Delft University of Technology

TU Delft Delft University of Technology

Abstract

Creating games is always an exciting occupation. In ten months, we have created a game for the Microsoft Imagine Cup, revolved around the Millennium Development Goals. Set on a distant planet, the game was to convey the MDG message in a more subtle way than most games did, but still recognizable. The most interesting gameplay mechanic in the game was the ability to travel back in time up to a maximum of five minutes, or until the energy meter runs out. This was also the most complex mechanic, and took the most time to implement correctly. In the end however, we prevailed and delivered a great game using a great structure built using a great engine. Unfortunately, the jury of the Imagine Cup did not seem to think so, and we did not make it into the finals.

Preface

This report is the result of one of the most unique Bachelor Projects at the Delft University of Technology. The goal of the Bachelor Project was for students to put into practice the Software Engineering skills and methods learned in previous courses, in a small team. The project spanned from September 2008 to July 2009.

The project consisted of creating a game to enter in the Microsoft Imagine Cup, an international multidisciplinary competition. The game had to revolve around the Millennium Goals, eight goals set by the United Nations to make the world a better place before 2015[1]. Besides wanting to help Microsoft raising awareness for these noble causes, we were also battling for a trip to Egypt and claiming the first place! This project has been done in collaboration with Cannibal Game Studios.

First of all we would like to thank Rafael Bidarra, without his enthusiasm for games this project would not be possible. We also would like to thank the people from Cannibal Game Studios, for providing us with great tools and being a great support during this project. Without them it would have been much harder to create a game of this quality. We are also grateful to David Scheper, for helping us out with modeling and helping to bring the world of Xylos alive. Finally we would like to thank Microsoft, for organizing such a great competition!

Paul Brussee
Nick Kraayenbrink
Bas van Sambeek

Delft, June 2009

[1]http://www.un.org/millenniumgoals/

Summary

During the last ten months, we have designed and developed a game for the Microsoft Imagine Cup. This game, built on the Cannibal Engine, was to revolve around the Millennium Development Goals. The obvious goal was to win the Imagine Cup, or at least to get as far as we could possibly can. Our team consisted of three programmers with little to no game design experience, and in the last couple of months a 3D artist helped us with creating the models of the game. In the end, we did not win, nor even made it to the finals, but we were still an experience richer.

The design of the game did not go as quickly and as smoothly as expected, but in the end we came up with a design everyone was happy with. It would be a game that did not incorporate the Imagine Cup theme as explicitly as most other entries of the year before, but still would convey the message of the Millennium Development Goals. Set on a distant planet, we also gave ourselves some artistic freedom with the character design, which was necessary because we did not have an artist yet. The main gameplay element of the game would be the ability to travel back in time, whenever for however long you wanted. In the end the mechanic became limited in its use by an energy meter that depletes when you went back in time, and by a five-minute limit of the time you can go back in time.

Implementation-wise this mechanic proved to be quite a challenge, especially in combination with an adventure-puzzle-platform game with enemies and other moving and changeable objects. However, with three programmers working almost full-time on it, any problems that arose were dealt with. Having three programmers also meant a sound and solid structure, except when the deadline nears, which is when anything goes, as long as it works. This did not have to happen very often, however.

Near the end of the development process, we got the Cannibal Composer from Cannibal Game Studios, to go with their Engine. Using this Composer the world was created, story elements and puzzles were added to the world, and all just in time for the deadline of round 2. Or rather a trifle too late, since the end result could have been that much better, even with just a few hours more.

All in all this has been a hard, but challenging and rewarding Bachelor Project, and none of us have regretted our choice of doing this.

Glossary

Adventure game A video game genre. In adventure video games (‘adventures’), the player controls one or more characters throughout the game, while being led through an interactive story driven by puzzle-solving and exploration.

Formal Grammar A tool in computer science to generate strings with a certain syntax. Consists of an alphabet, a set of symbols disjoint from the alphabet, a start symbol taken from the alphabet and a set of production rules. Is also called a ‘rewriting system’.

FPS Frames Per Second. The rate at which the host machine can render frames.

Frame Either: one visual rendering of the game world by the host machine. Or: the timespan between two consecutive renderings of the game world by the host machine.

Framerate See: *FPS.*

HKU ‘Hogeschool voor de Kunsten Utrecht’. A college for various forms of art, based in Utrecht and Hilversum.

HUD Head-Up Display. A method by which information is visually relayed to the player as part of a game its user interface.

Low poly A term indicating that a 3D model is made of a relatively low amount of polygons.

MDGs Millennium Development Goals. The Millennium Development Goals are eight international development goals that the United Nations have agreed to achieve by the year 2015. They include reducing extreme poverty, reducing child mortality rates, fighting disease epidemics such as AIDS, and developing a global partnership for development.

3D Model A mathematical wireframe representation of any three-dimensional object. Usually a texture is applied to the model. The wireframe can be defined by an arbitrary amount of polygons, or by an arbitrary amount of curves.

NPC Non-Playable Character. A character in a game world that is not controlled by the player.

Platformer A video game genre. The main activity in platformer games (‘platformers’) is usually jumping to and from platforms and over other obstacles.

Polygon A polygon is a plane figure, composed by a finite sequence of straight lines. Every endpoint of every line should connect to an endpoint of another line.

To respawn To reappear from the same place.

Contents

Preface i

Summary ii

Glossary iii

I Introduction 1

1 Introduction 2

2 Introducion to the Imagine Cup 3

II Design 4

3 Story design 5

3.1 The rough shape . . . 5
3.2 Creating a proper story . . . 5
3.3 The result . . . 6

4 Gameplay design 7

4.1 Genre . . . 7
4.2 Six Dimensions of Fun . . . 7
4.3 Impossible Adversaries . . . 9

5 Level Design 10

5.1 Shapes . . . 10
5.2 Map Layout . . . 10
5.3 Puzzles . . . 11

III Implementation 14

6 Process 15

6.1 Quest for the artist . . . 15
6.2 Partnership with Cannibal Game Studios . . . 16
6.3 Scrum . . . 17
6.4 Dealing with deadlines . . . 18
6.5 Internal Process . . . 18
6.6 Testing . . . 19

7 Structure **20**
7.1 Game Content & Logic . . . 20
7.2 Namespaces . . . 20

8 Game States **25**
8.1 InGame State . . . 25
8.2 CutScene State . . . 25
8.3 Loading State . . . 26
8.4 Menu State . . . 26

9 Time Travel **27**
9.1 Storage Requirements . . . 27
9.2 Time Requirements . . . 28
9.3 A Point . . . 28
9.4 A Line . . . 29
9.5 A Split . . . 30
9.6 A Fold . . . 31

10 Trees **34**
10.1 L-systems . . . 34
10.2 Implementation of the trees . . . 35
10.3 Integrating time travel in trees . . . 37

11 Use of the Cannibal Composer™ **38**
11.1 Composer Converter . . . 39
11.2 Difficulties . . . 40

IV Evaluation **41**

12 Reflection **42**
12.1 Reflecting on the process . . . 42
12.2 Reflecting on the product . . . 43

13 Conclusion **44**

14 Recommendations and Future Work **45**
14.1 Recommendations for the BSc Project . . . 45
14.2 Recommendations for future participants . . . 45
14.3 Into the future . . . 46

V Appendices **47**

A TreeTurtle character mapping **48**

B ‘Triads’ **49**

C Plan of Approach **50**
C.1 Introduction . . . 50
C.2 Project contract . . . 51
C.3 Approach and planning . . . 52
C.4 Project organization . . . 53

C.5 Quality assurance 53
C.6 Appendices 54

D Orientation Report **55**
D.1 Story and Setting 55
D.2 Style 56
D.3 Techniques 57

E Imagine Cup Deliverables **61**

Bibliography **73**

Part I

Introduction

Chapter 1

Introduction

Video games are popular with people of all ages and interests, mainly because of the large variety of games. A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a video device. Video games have been around since the 1940's since people started playing games on the 'Cathode Ray Tube Amusement Device'[1] and since then video games have grown more and more in popularity. We all have been avid gamers since we were young and started appreciating the art of video games even more after our experience with creating a game during the MKT4 project.

When we first heard about the opportunity to create a game as Bachelor Project we were all interested and were confident that we could create a better game as we did in the MKT4 project with our gained experience. Obviously this project is not like any of the regular BSc projects which makes it even more interesting. Part of this BSc project is competing in the `Microsoft Imagine Cup`, of which some background can be found in chapter 2.

Another difference would be be the time frame available in which the product (the game) needed to be developed. The deadline for Round 1 was March 1st 2009, so starting as late as the second half of the second semester would not have been an option. Instead, actual development already started on January 1st 2009, while the brainstorming and design phase had already begun a few months before. While the deadlines of the *Imagine Cup* needed to be upheld, it was by no means a requirement to continue to the next round, as that decision was up to an independent jury.

The rest of the report is divided into three parts; Design, Implementation and Evaluation. Part II, Design, describes mostly the first part of the project: designing the game. Then Part III, Implementation, describes the second part of the project: implementing the game. In reality these parts were not so cleanly separated, and some things had to be redesigned multiple times after noticing deign flaws while or after implementing them. Finally, Part IV, Evaluation, contains our evaluation and view on the project, and how to continue in the future.

[1]http://en.wikipedia.org/wiki/Video_game

Chapter 2

Introducion to the Imagine Cup

The Imagine Cup is an annual global student technology competition focused on finding solutions to real-world issues. Since 2002, the Imagine Cup is organized by Microsoft. We competed in the Game Development competition and the the objective was to create a new game that uses both Microsoft's XNA Game Studio 3.0 and Visual Studio 2008. The game had to illustrate the Imagine Cup theme of this year: "Imagine a world where technology helps solve the toughest problems facing us today." The United Nations had identified some of the those tough problems today in its Millennium Goals. These goals were agreed upon by 189 nations around the world more than eight years ago. They encompass universally accepted human rights such as freedom from hunger, the right to basic education, the right to health, and a responsibility to future generations. The target date by which the Millennium Development Goals are to be achieved is 2015. Our goal is to create a game that is about these goals.

The competition also offers a chance to meet the other students from around the world and a chance to win cash prizes, internships at Microsoft, and even a free trip to Cairo, Egypt if you proceed to the Worldwide Finals.

The competition started on August 29, 2008, and ends on July 7, 2009. There are three rounds and each round you have to submit certain entry deliverables. We started off with about 750 teams globally in round 1, which was reduced to about 150 in round 2. For the last round, the worldwide finals, only 6 teams were selected to advance to compete in Cairo, Egypt from July 3–7, 2009.

Part II
Design

Chapter 3

Story design

The story of the game did not fall from the sky. As can be read in the Orientation Report in appendix D, we thought of several different stories and settings for the game we wanted to make.

3.1 The rough shape

How exactly did we come up with the story about Xylos and his buddy Kron? Because we wanted to deliver the serious Millenium Development Goals to a young audience, we decided we should not make it very realistic. So no children in Africa living in the slums or other terrible images. We decided to go for a world that was not earth, but had similarities. By doing this we had some artistic freedom on how to deliver the message and this would provide us with a platform to deliver the Imagine Cup goals in a non shocking, and hopefully even cheerful way. This would also make it more easy on the graphics side, because if we wanted to create a realistic looking game, small flaws would be much more notable.

Because we are all avid fans of 3D platforming titles, like Ratchet & Clank and Jak & Daxter, we wanted to create a title just like that. During a brainstorm meeting one of us noted it would be cool to have a time traveling device so we could manipulate time, creating an unique gaming experience. We knew there existed 2D games that would let you manipulate time, but we could not find any games what would do the same in a 3D environment. And since this is a Bachelor Project, we decided we would accept that challenge!

We started working with those two ideas and made a list of the Millennium Development Goals to see how we could combine those with the two ideas. Soon we came up with the idea that our main character was being suppressed by an evil ruler. The character then would need to work for the ruler in exchange for food and water.

Besides just making a game centered about the Millennium Development Goals, the Imagine Cup also had a sub-theme: *'Imagine a world where technology helps solve the toughest problems facing us today.'* Therefore we thought it was cool to let the technology that makes it able for you to travel in time, would be a gadget, something silly like a pocket watch. And to make the technology even better, we wanted to let the technology be alive, so it could talk. This made us able to kill two birds with one stone. It would be fun but most importantly it would provide a cool way to tell the story. Since the time gadget had been around on the planet for a while, he could tell the player what happened and how the evil ruler took the throne. Consequently, the gadget could also tell how he should be used. The idea was then to let the player go back in time and stop the ruler before he took the throne.

3.2 Creating a proper story

Once we made up the ideas mentioned in the previous section and created our core idea, it was key to fit it into a proper story. We started doing so by creating names for our different characters, mainly because it would be easier to talk about when discussing ideas. It is more easy to refer to 'Xylos' as to 'the guy that is being suppressed'. We came up with the name 'Xylos' because for some odd reason one of us was thinking about a xylophone when we where thinking of names. It was supposed to be a temporary name, in the hope we would find an artist who would come up with a better name, but we grew accustomed to it and never changed it. Same story goes for our planet, called Warf, with evil ruler Worf. The name of our gadget was a harder nut to crack, but after a while we settled on Kron[1].

Because being suppressed was not a MDG, we tried to transform three other goals into a gadget. We chose food, water and health, mainly because we found them likely to be fixed by a gadget. The food problem would be fixed by a rapid food grower, water by a cleaning system and health by a vaccination gadget. These gadgets would be created by *the ancients*, the ancestors of Xylos. The ancients were a clever civilization, always inventing new tools and hardware and living in harmony. Besides those gadgets we tried to incorporate education as well. Because Xylos and his fellow Warfians were surpressed for all those years, they never leaned to read. The player would then need to find books scattered around the level to learn the people how to read.

3.3 The result

On a day Xylos, a young boy who is the main character of this story, finds a strange device in the sand while he is working. Xylos and his fellow Warfians are being suppressed by Worf, ruler of the planet Warf. Worf has taken away all the resources from the people so that the people are dependent on him. All the Warfians are working for Worf in order to get some food and shelter. The device Xylos finds turns out to be a time traveling gadget called Kron who tells our hero how Worf took the throne a long time ago. Apparently Worf seized the food gadget, enslaving the ancients because now he had control over an important factor of life and the ancients now relied on Worf for food. Together Xylos and Kron decide to prevent this from ever happening and travel to the past to find the first gadget, Deme[2]. By grabbing Deme before Worf could, they hope Worf could not enslave them and they head back to the present.

Back in the present they start looking for seeds, because the tree gadget needs something to start with. During their quest for seeds they notice that suddenly all the water has been polluted! Not knowing how this happened, they soon there after find an ancient trapped in between a pile of rocks. After saving her life, it turns out to be Marcella Ploo, a teacher whom you have met in the past. She tells you that she is the last ancient on the planet and somehow managed to kept herself alive. Xylos tells her about his quest and asks her about the polluted water. Ploo thinks that because he took Deme, Worf now has managed to grab Osei[3], the water gadget. Unfortunately she does not know where or when Osei is, so Xylos starts looking for seeds again. During his search Xylos finds a strange gizmo called a book! Curious of what it is he takes it to Ploo. Ploo is delighted to see the book and tells Xylos that if he sends some children to the school, the children might be able to find the location of Osei in those books! When Xylos has sent enough children to school and gathered enough books, Ploo will tell him the location of Osei and Xylos and Kron head back to the past.

When they have gathered Osei and return to the present, it would basically be the same story all over again. Xylos will clean the rivers and finds his fellow Warfians now being very ill because Worf now has taken Acea[4], the medicine gadget. When the children have found the location of Acea, Xylos travels back to the past again, and heals the people when he comes back. After collecting all the gadgets Xylos decides enough is enough. He heads back in time one final time to deal with Worf once and for all in an epic boss battle to free all Warfians from Worf's oppression!

[1]Named after Chronos, the Greek personification of time
[2]Named after Demeter, the Greek Goddess of grain and fertility.
[3]Named after Poseidon, the Greek God of the sea.
[4]Named after Panacea, the Greek Goddess of healing

Chapter 4

Gameplay design

A game is nothing without gameplay, and it would be more like a movie if it did not have any. This chapter focuses on several aspects of the gameplay of our game. Section 4.1 briefly discusses the genre of the game. Then the time travel mechanic is discussed in section 4.2. Finally, enemies spark the discussion in section 4.3.

4.1 Genre

With the rough shape of the story we almost automatically assumed the game was to be a platformer. Later, during the course of the design, we unconsciously thought of more and more elements that had more to do with an adventure game. We kept calling it a platformer, however.

With the time travel gameplay element we really wanted to incorporate in the game, we did not really have a choice. The other option available would be a puzzle game, but the notion of making a pure puzzle game was quickly dismissed. None of us really felt anything for a pure puzzle game, and the notion of a hybrid puzzle-platformer or puzzle-adventure was never mentioned.

It was only after the first deadline, during a meeting with the guys from Cannibal, that we were set into thinking about the genre again. Because we still did not have an artist at that time, and because of the limited time still available for the second round, it was advised to us to lean more into the direction of a puzzle game, instead of keeping it adventure-platformer. We realized this was true, but we still wanted to make one world with puzzles, instead of creating a separate area for each puzzle (and having loading screens, however short, between them). We have not deviated from that course, as far as we are aware, up to the final deliverable for round 2.

4.2 Six Dimensions of Fun

Once mentioned, it proved to be impossible for the time travel mechanic not to make it into the game. All of us thought it would be great, and all of us thought we could pull it off. And we did pull it off (see chapter 9 for implementation issues and details), but only after we designed the mechanic and its limitations. In the end, we made one of the first six-dimensional games (see appendix B for more details).

4.2.1 General behavior

We had seen a couple of games that already made us of some sort of time travel mechanic. One of them was *Braid*[1]. While certainly an interesting and well-made game, we did not like that fact that in most levels the actions you did before going back in time had no effect. And in the level where you did have a 'time-clone',

[1]http://braid-game.com/

the maximum of ‘previous selves’ was only one. Our desire was to practically not have a limit to how many previous versions of the player were being replayed.

Another game we looked at was *Chronotron*[2]. If we got any inspiration from this game it was for the puzzles, as the mechanics were not what we envisioned. In *Chronotron*, you have to get back to where you came from, a time machine. Also, you could not partially rewind time, but only start again from when you exited the time machine. These were both things we did not want to have. The player would have his ‘time machine’ always with him, and would be able to go back to a practically continuous range of moments in time (as was the case in *Braid*).

4.2.2 Limitations

What we did realize was that we could not afford to allow the player to rewind as much as he desired. One aspect was that the memory requirements for such behavior would soon become too large. Another would be that, because the game was going to have some kind of interactive story, the player should not be able to go back in time to before moments story progression was made.

As is discussed in chapter 9, in the end the time one would be able to go back was not limited because of the techniques used, but because of a hard-coded limit. Any story progression was also to be permanent, and dialogs with NPCs would be ignored when traveling back in time.

However, with these limitations, it would still be possible to flood the memory with a lot of previous versions of the player. Therefore we added an energy meter as extra limitation to time travel. If the meter was depleted, one would not be able to go back in time any further. On top of that, the meter decreased faster if the amount of previous versions of the player increased. This way the amount of times the player could go back in time did not have a theoretical limit, but it would have a practical limit[3].

4.2.3 Other considerations

We had learned something from the game *Chronotron*, and that was that differentiating the player from his previous versions can be difficult if there are too many. It would happen less in our game, since the current version of the player would always be in the center of the screen, but we still decided that the previous versions should look different. Changing the color was out of the question, as you still had to see instantly that it was a previous version of the player, and not some random thing popping up and doing seemingly random things. This was one of the reasons to make the previous versions of the player semi-transparent.

The other reason was that we decided one would not be able to collide with the previous versions of yourself. We chose to do this, because it would be hard to force the player out of its previous version when time started flowing forward again. If the player stood in the wrong place (eg: next to an abyss), and the player was forced out in the wrong direction (eg: into the abyss), the possibility exists that the player would be stuck there forever.

We also wanted to avoid as many paradoxes as we could. Therefore we decided to make any object a previous version of yourself interacted with unalterable until that previous version disappeared. Additionally, any collectible item (eg: seeds, energy rechargers) a previous version of the player will pick up should be made transparent a well. This is to indicate to the player that this object can not be picked up again.

Initially, we used collectible energy recharges that did not respawn. However, because this could mean the player could get stuck at some point because the energy meter was empty and all energy recharges were gone, we decided to use respawning energy recharges. We did reduce the amount of them however, and now they are usually only placed at the start of a puzzle.

[2] http://www.kongregate.com/games/Scarybug/chronotron

[3] It would not have a limit if enough respawning collectible energy recharges were placed close enough to each other

4.3 Impossible Adversaries

It was quickly apparent there would be enemies in the game, but not in what form, or even if they would appear in the part of the story we were going to make. The difficult part was designing the enemies in such a way that you could use, or had to use, time travel when encountering the enemy. On top of that they had to fit the style of the game.

4.3.1 Death

But before we could think about incorporating enemies into the game, we had to think about how to handle the death of the player. Because you have a time travel gadget that can think for itself and activate itself when necessary, one may expect that he rewinds time so that the player is alive again. This is exactly what we did.

However, the question of how far back in time to go remains. Preferably, the player can decide for himself how far back in time to go. However, this would mean the energy bar would have to deplete as well, to prevent the abuse of this mechanic to go as far back in time as desired. Alternatively, the player would travel back in time for a fixed amount of time without depleting the energy meter, but how far back in time would be a good amount?

We also had to take into account that the player could fall into ravines or other deep pits, so the solution would have to work for those kind of scenarios as well. In the end, we chose to make the rewinding happen automatically. After some testing, we settled on forcing the player go back five seconds, which did not feel like being a too long time when playing. The only thing we had to keep in mind was that the player should die before falling down for more than five second, but this posed no problem.

4.3.2 Enemies

Initially, we thought of various ways we could incorporate enemies into the game. For example, they would have to be hit from the back to kill them, but they would approach the player once in sight to try to kill the player. Or they would shoot at the player from a stationary position, but a hit from any side would kill them. Or they would simply be stationary things that would only kill if you touched them, forcing you to take another route.

However, for a long time the priority of implementing enemies was quite low on the list, as we were concentrating om improving the already present gameplay elements and designing the puzzles. After a while we got the idea that implementing an enemy would not have to take a lot of time, if we used part of the structure we had already built for other purposes. We decided we could at least try, and implemented the enemy that followed you when it was within his visibility range.

When the first version was done, a flaw in the design quickly became apparent. Because the enemy would do very little if the player could run faster than the enemy, we made him slightly faster. However, we did not anticipate it would be a problem that the enemy could go everywhere the player could go. When an enemy got sight of the player, he would run after him, and eventually kill him. Then time is forcefully rewound five seconds, after which it would oftentimes occur that the player was in the path the enemy would take, because of the time it took for the enemy to gain in on the player.

We fixed this in two ways. The first being that we made the enemy faster, but still not too fast, to prevent instant death when the player is spotted. The second was that we chained the enemy to an anchor, so that it could not follow the player anywhere. This anchor also became the thing that would 'kill' (or rather 'deactivate') the enemy. By interacting with the anchor, the energy that feeds the enemy is cut off, rendering the enemy unable to do anything (not even kill the player when the player touches the enemy).

Although we would have liked to, we did not have enough time left for round two to implement different kinds of enemies. This might even be for the best, to prevent frustration about the fact that two of the four enemies that fin in the game are different from the first two encountered.

Chapter 5

Level Design

When we had finally decided what the game would be about, we had to design the location where all the action would take place: the planet Warf. Designing the planet Warf was quite a challenge since we are not graphics artists and did not manage to find one when we had to start designing it. From the start it was clear we needed a large world, because what is a 3D platform game where you are not able to walk around.

5.1 Shapes

Because we were not able to find a graphics artist at the time, we had to make a tough decision. Our ultimate goal was to create a realistic looking game, but due to the lack of artists we were forced to a design which still would look fun and attractive to young players, but with the benefit that we could actually make it ourselves in Autodesk® Maya® with our limited design skills. Since Maya lets you create models with basic geometric objects as a starting point, we decided that was the way to go for us.

We decided it would be fun, and easy for us to model, to let each part of our village be a basic geometric shape. Since a triangle looks sharp and more aggressive it was clear that would be the shape of the enemy, whilst our hero and his fellow people would look nice and friendly, like a sphere. Because we wanted to make a clear distinction of the suffering the Warfians had had over the last few years, we decided to let the older Warfians from 1337 years ago have a square, more robot like shape, and the Warfians had eroded you might say into a rounder shape over the years. Because we wanted the shapes to play a key factor in the game we wanted to incorporate the geometric shape theme into the complete game, even making round fences and paths.

5.2 Map Layout

When we had settled on the shapes, it was time to draw the map of the planet Warf. We decided it was best to start sketching the map of Worf that was staged in the present, since the world of the past would look roughly the same structure wise, but might look a bit darker graphics wise. So the map of the Warf of the past would just have fewer buildings that did not exist back then.

As you can see on figure 5.1 there are three distinct area's. The top right round one is the village where Xylos lives after work with his fellow Warfians. They started living here after they were banished from the square village where they lived before Worf took the throne. The square village is located at the bottom left and features box like houses that by now have turned into ruins. Because it has been abandoned by the population it looks a bit like a ghost town. In the center we find the castle of Worf, our evil ruler. The castle is surrounded with a fence so Worf can keep out those annoying Warfians (atleast, he thinks they are annoying...). To the right of the castle you see an area that is called the WWW. No, that does not stand for World Wide Web but it means *Worf's World of Work*. Into tricking the Warfians that working for Worf would be fun, Worf devilishly gave the stone mine a cheerful name. From the WWW Xylos needs

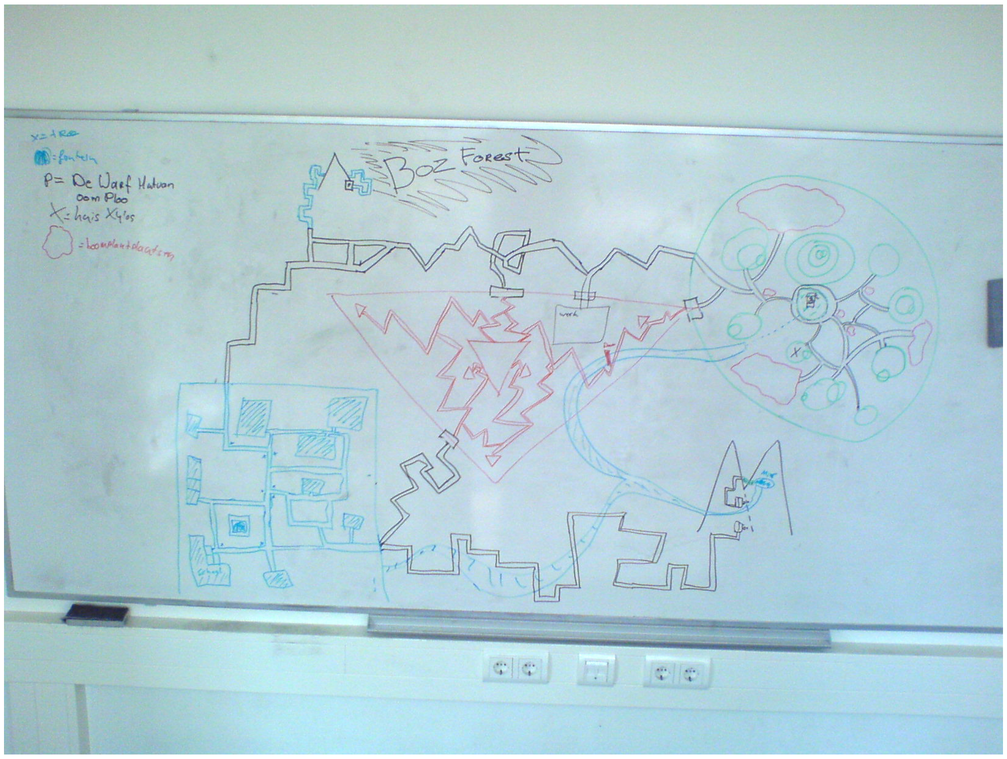


Figure 5.1: Layout of the map we doodled on the whiteboard.

to escape to the tower next to it, the Time Travel Location. The castle area features three of those Time Travel Locations, each located in a corner of the triangle, and only in those places can Kron travel far back in time.

We decided it was best that traveling back in time was limited to a few fixed locations in the game for a few reasons. By doing so we could let the story lead to gaining access to this tower, so a part of it was story driven. The other reason was game play: if had let people travel back in time whenever, wherever they wanted, it would have been much harder to structure the storyline and would make the game less fun.

As you can see we kept the 'Geometric shape'-style even in the road design to emphasize the shapes of the people living there, and even the houses are based on the shapes belonging to the respective area. Everything you will see is related in one way or the other to the shape.

5.3 Puzzles

A real challenge for us was how to limit the usage of the time travel device, Kron. At first we wanted to keep true to the real 3D platformer spirit and just put time trails on the path of our different levels, but we soon realized this would lead to smaller puzzles and does not give a clear distinction on when you need to use what. After a bit of messing about with the time mechanism we found that the best way to keep it fresh and innovating every time was to incorporate it in different puzzles to give the challenges where you need to manipulate time a clear start and finish.

But that gave us a new challenge: in normal puzzle games the puzzles are selectable trough a menu and you just play the puzzles after you have selected one, but we planned to fit them in a 3D platformer! To solve this problem we wanted to cluster the puzzles. So you would walk into a open space in, for example, the woods and there you would see 5 different doors. Behind each door there would be a puzzle the player needs to solve. Clustering the puzzles would bring a few advantages. We could let each cluster be of a different difficulty level, for example puzzles 5-10 would be of level ‘Easy’ and cluster 11-15 then could be difficulty level ‘Medium’ and this would give a clear level of distinction for the skills required for the puzzles. It would also provide us with an option that we could let the player skip a level and only play 4 out of the 5 levels if one proved to difficult. By providing this option there would be fewer chance for a player to get stuck on a level and quit the game. Now he could finish the game and maybe try the puzzle again later after giving it some thought.

The goal of each puzzle is to get to the location of a seed. The seeds are necessary to feed the population of Mellut, your home town.

Descriptions of the puzzles

We planned to start with 5 mandatory puzzles at the beginning of the game that had to be played in order instead of cluster wise as some sort of tutorial puzzles. During these puzzles Kron would talk a bit more then usual and guide you through them to familiarize the player with the controls and the different elements a puzzle could consist of. Because we wanted to start really simple we started with

Puzzle 01: The One
‘The One’ is the most simple puzzle there can be, the player needs to step on a button. Stepping on this button makes a forcefield shut down, allowing the player to continue on his path.

Puzzle 02: We’re closed
‘We’re closed’ is almost the same as ‘The One’ but with a twist, whenever the player leaves the button the forcefield will be reinstated. This is to bring in the time travel aspect of the game. The player needs to stand on the button for a short while, go back in time and then quickly walk past the forcefield, whilst his previous version still stands on the button.

Puzzle 03: Going Up
In ‘Going Up’ the player sees a platform and a button on the ground before him. When the player stands on the button the platform will start ascending, but if the player leaves the button to jump onto the platform, the platform will start to descend. So by manipulating time again, the player needs to stand on the button to elevate the platform to its maximum altitude, rewind, and quickly jump on the platform whilst his former self is standing on the button.

Puzzle 04: Seeing Double
‘Seeing Double’ is again a variation of the first puzzle, but now you need to step on two buttons at once. After the player has managed to step on two buttons at the same time by manipulating time, a forcefield will shut down, and stay that way.

Puzzle 05: Double Lock down
Like puzzle two, ‘Double Lock down’ is a step more complicated as its predecessor. Like the previous puzzle this one also features two buttons and a door, but this door will shut down after one of the button has been released. Similar to puzzle two, this requires the player to create three instances of himself for the first time.

Level 5 concluded the tutorial section, after that we designed a first cluster.

Puzzle 06: Up Up And Away
Puzzle six ‘Up, Up And Away’ consists of three elevator platforms and three buttons, but with a twist. In order to reach the second platform, you need to elevate the first platform, and in order to reach the third, you need to elevate the second. So before you climb up any elevators, you need to elevate them all at the beginning and then you will be able to climb to the top.

Puzzle 07: Combination is key
This puzzle consists of a door with tree buttons. You might say, 'I need to stand on all buttons at once' but that would be wrong. To open the door you will need to find the combination of two different buttons that will open the door. Two buttons need to be pressed at once, leaving the other untouched, and the forcefield will be shutdown.

Puzzle 08: Sinusoid
You are faced with a ravine and in order to reach the other side, you'll need to bring up the platform laying on the bottom of the ravine. The tricky part here is to land exactly on the platform after you have jumped, because if you miss the platform you will land on the bottom of the ravine and end up dead. And we would not want that, now would we?

Puzzle 09: The Platforms to the Tower Go Round and Round
The seed is located on top of a tower and you can reach it with a moving platform. The platform will first go to three lower towers, also containing seeds. But you will only get one shot, because after the platform has set off, it does not come back! (Except when you rewind time of course…) This means you will have to time your jumps on and off the platforms to collect the apples and maybe rewind if the platform has traveled too far whilst you were reaching for the seed.

Puzzle 10: The One That Has Not Been Made
Unfortunately we did not have time to design the last puzzle we wanted to hand in for round two.

Part III

Implementation

Chapter 6

Process

6.1 Quest for the artist

When we signed up for this bachelor project we formed two teams of three students, all with a coding background. But because a game consists of more than technical bits, it needs to look appealing for the player for example, we started looking for a fourth team member. So unfortunately, due to group limitations of the Imagine Cup, we could only add one artist to our registered team. While this is not much for any game, however good he or she might be, it certainly is not for a game with the scale we wanted to make. But since no artist at all would be more troublesome, we started the search by contacting the students of the HKU with whom we participated in the MKT4 project last year, since it's always nice to work with people you already know and trust.

Unfortunately the third year of the HKU seems to be quite busy with their regular subjects and internships, so none of the people we knew had the time to work with us. But we did get in contact with a student who was willing to help us and through him we were able to send two mails to all the second and third year students of the HKU.

Meanwhile we were also busy looking for people closer to home. With thanks to our supervisor, Rafael Bidarra, we found a teacher of the faculty of Industrial Design who was teaching a graphics course and was willing to let us give a short presentation at the beginning of one of his lectures. Unfortunately the students did not look very interested and it was also all quiet at the HKU front. Because we already anticipated this a bit, we already started designing the game by ourselves. This process is mentioned in chapter 3: Story Design.

Fortunately we got a mail from Microsoft, saying that they were willing to mail a short description of the game we had made up to some colleges and universities they visited with the Imagine Cup Academics Tour. With renewed hope, we provided them with a 2 page summary of our game, hoping we would get an artist out of this, since it is more easy to commit to something if you know what it is about. But like all of our previous endeavors we ended up with nothing.

By now it already was the 20th of March and we already had submitted our first entry. Apparently some more teams were struggling with finding teammates, because we got a mail from the Imagine Cup, saying they would give all the teams one more weekend for adding more members to their team and after that the deadline was closed. So as a last Hail Mary pass we went to the Grafisch Lyceum in Rotterdam, hoping they could provide us with our most wanted artist. But even here the answer was the same, the first year students were rendered to be too inexperienced to work alone in our team, and the higher year students were too busy with their own projects. So despite our efforts we ended up with no artist at all.

But then there was hope! Just a month before the deadline of round two, Paul came in contact with someone who knew some graphics artists and she found someone who was willing to join our team! Because we only had one month left it would be challenging, but since we decided to go for a low poly design, consisting of mainly basic geometric shapes, it was feasible. And after eight months of trying, we finally had an artist!

6.2 Partnership with Cannibal Game Studios

When we started with the project we got a mail from Remco Huijser of Cannibal Game Studios. He heard that we were going to make a new game in XNA and said that he and Cannibal Studios might be able to help us out with their Cannibal Engine and Composer.

6.2.1 The Cannibal Engine

The Cannibal Engine is a platform built on top of XNA that powers our game. It for example contains easy tools for Scene Management, Graphics, User Input and User Interface. In the MKT4 project we also had worked with the Cannibal Engine, so we knew it would help us with development of the game.

Although the Cannibal Engine is based on XNA, it is quite different. Because the Cannibal Engine is built on top of XNA, we would have little to do with XNA itself, which is convenient most of the times but also came with a few downsides. For starters we could not access XNA directly, which also provided some problems. For example, XNA has a large user community which can be used for support and examples. So if we found an algorithm or fancy graphics shader we could not use it, whilst we were both using XNA. This meant we had to do everything on our own, with some help of Cannibal of course.

Another downside was that the engine was, and still is, an unfinished product. Some features, like particles and post-processing effects, would still be missing initially. Also, if we experienced any bugs we had to think twice if they were caused by a problem on our end or by a problem caused by the engine. But in a way that provided an extra challenge for us, whilst creating a game we would also be the guinea pigs for Cannibal since we would be putting their platform under severe stress.

6.2.2 The Cannibal Composer

Besides the engine Cannibal is also working on a different product called the Cannibal Composer. The Cannibal Composer allowed us to create a large world really fast. Normally we would have needed to give precise coordinates of where our objects would need to appear in the world, but with the Composer we could just drag and drop things. It also allowed us to sculpt the terrain of the worlds to create nice height maps for Xylos to walk on. This provided us with a much better and precise option as opposed to how we handled it before: drawing a height map in Paint Shop Pro X with gray scaled colors.

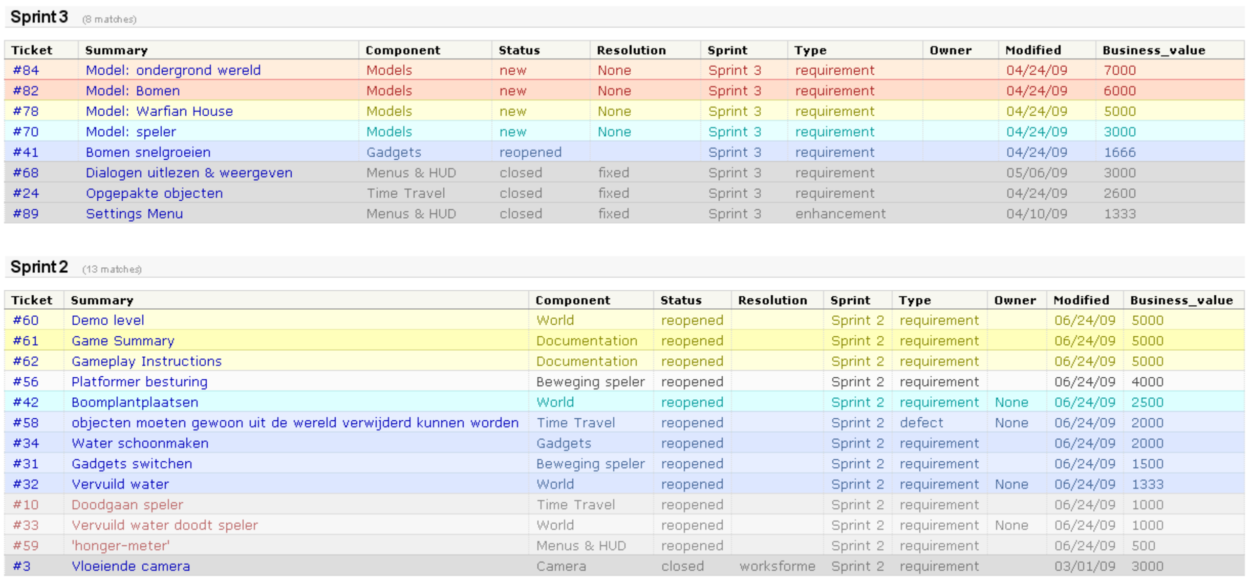
6.2.3 Meetings

Besides four team members an Imagine Cup team can also contain a mentor. The mentor of a team is usually an experienced Industry Professional who will guide the team through the process of making the game and support them where needed. Because we would have some contact with Remco Huijser of Cannibal Game Studios anyway, since we would be using their engine, we asked him if he would be interested in mentoring out team. Fortunately, after some deliberation he said yes and agreed to help us with structuring the process and game tech, but to ensure we all took the project seriously he had a few demands:

- At least once a week contact about the progress of the project (be it by mail or telephone)
- We meet every two weeks face-to-face
- We are actively involved in evaluating the Cannibal Products, so we submit bugs, ideas etcetera.

Of course these were demands we were happy to comply to, because it would benefit us as well.

During the meetings we usually discussed our progress, like how are things going with regards to finding an artist, and what kind of new features we would like to see in the engine. Unfortunately, near the end of the project Remco got caught up with various other activities for Cannibal, so he would not have any time



Sprint 3 (8 matches)

Ticket	Summary	Component	Status	Resolution	Sprint	Type	Owner	Modified	Business_value
#84	Model: ondergrond wereld	Models	new	None	Sprint 3	requirement		04/24/09	7000
#82	Model: Bomen	Models	new	None	Sprint 3	requirement		04/24/09	6000
#78	Model: Warfian House	Models	new	None	Sprint 3	requirement		04/24/09	5000
#70	Model: speler	Models	new	None	Sprint 3	requirement		04/24/09	3000
#41	Bomen snelgroeien	Gadgets	reopened		Sprint 3	requirement		04/24/09	1666
#68	Dialogen uitlezen & weergeven	Menus & HUD	closed	fixed	Sprint 3	requirement		05/06/09	3000
#24	Opgepakte objecten	Time Travel	closed	fixed	Sprint 3	requirement		04/24/09	2600
#89	Settings Menu	Menus & HUD	closed	fixed	Sprint 3	enhancement		04/10/09	1333

Sprint 2 (13 matches)

Ticket	Summary	Component	Status	Resolution	Sprint	Type	Owner	Modified	Business_value
#60	Demo level	World	reopened		Sprint 2	requirement		06/24/09	5000
#61	Game Summary	Documentation	reopened		Sprint 2	requirement		06/24/09	5000
#62	Gameplay Instructions	Documentation	reopened		Sprint 2	requirement		06/24/09	5000
#56	Platformer besturing	Beweging speler	reopened		Sprint 2	requirement		06/24/09	4000
#42	Boomplantplaatsen	World	reopened		Sprint 2	requirement	None	06/24/09	2500
#58	objecten moeten gewoon uit de wereld verwijderd kunnen worden	Time Travel	reopened		Sprint 2	defect	None	06/24/09	2000
#34	Water schoonmaken	Gadgets	reopened		Sprint 2	requirement		06/24/09	2000
#31	Gadgets switchen	Beweging speler	reopened		Sprint 2	requirement		06/24/09	1500
#32	Vervuild water	World	reopened		Sprint 2	requirement	None	06/24/09	1333
#10	Doodgaan speler	Time Travel	reopened		Sprint 2	requirement		06/24/09	1000
#33	Vervuild water doodt speler	World	reopened		Sprint 2	requirement	None	06/24/09	1000
#59	'honger-meter'	Menus & HUD	reopened		Sprint 2	requirement		06/24/09	500
#3	Vloeiende camera	Camera	closed	worksforme	Sprint 2	requirement		03/01/09	3000

Figure 6.1: Some of our Scrum Sprints on the Cannibal Experience platform. Some tickets are reopened for illustrative purposes.

left for us. But no harm done, Nigel Karsidi, another Cannibal Employee, took his spot and from then on we had weekly face-to-face meetings with him. We increased the face-to-face meetings for two reasons: because we started to depend more and more on missing Engine features, we wanted to keep informed on how the status of the engine was. The second reason was that we were focusing more and more on the tech behind the game, that we were losing sight of the gameplay. Fortunately the Cannibal guys noticed that on time and Nigel started helping us think about the gameplay of our game.

6.3 Scrum

As described in our Plan of Approach, we used the agile development method called Scrum method as our development process. The Scrum method was advised to us by Remco, they use it themselves at Cannibal Game Studios for the majority of work, and said it would suit our kind of project. For more information of what Scrum is about, please look at the description we gave in the Orientation Report.

We started using Scrum in the first week of development by listing the requirements. Although not everything was clear about the game and all the little details had not been worked out yet, we listed our core features like the time travel mechanism. After the creation of the list we played the so-called ‘Business Value game’; the Business Value game is a way Scrum uses to rate each components importance. For this game we all had a set of cards with ratings going from 1 to 7000, one being not important and 7000 very important, and each player put their ratings on the table for a particular subject. If there was a high difference between ratings, we discussed why a person thought it was so important (or vice versa) and then settled on a score we all agreed with. If there was less difference between the ratings we settled on an average score. In figure 6.1 you can see a screenshot taken from our Cannibal Experience platform of our Sprint Backlog we created after playing the game. When the scores were set and the sprint defined, the development could start.

As you can see in figure 6.1 we ended with only three sprints. This is mainly due to the fact that our sprints took way too long, so they became more like a marathon instead of a sprint, and our end dates were not clearly defined. The sprint just ended when everything was done, which usually was when the next deadline by the Imagine Cup was set. If we had defined sprints of like two weeks time, the priorities might have shifted.

Another mistake we made is that we did not have a Scrum Master or Product Owner. The Scrum Master is a person who is responsible for the sprint and makes sure everyone does what they are supposed to do within the time limit, whilst the Product Owner is the person responsible for the product we will deliver in the end: the game itself. The way we worked was 'If you are done with your task, take the next task with the highest priority and start working on that one', and everyone was responsible for everything. Although we think we managed decent enough with managing responsibilities, we did not encounter any problems on that side, it might have been better if from the start of a sprint tasks were assigned to different people. By doing so you immediately knew what you were supposed to do and how much time it would cost you complete it.

Although we liked the way scrum works, sometimes it could be a bit redundant, so for the last deadline we took a different approach. Instead of incorporating all scrum methods, we took our backlog and then use a bubble sort on what was the most important. 'Do we think requirement A is more important as requirement B?', then we would put requirement A above the other. This proved to be a just as good solution for us, because whether it was more important by 3000 points or just 100, the one at the top would be developed first. But we can imagine some cases where the Business Value would play a bigger factor.

All in all, we still believe Scrum has much to offer for agile development and we believe we could have gotten more out of it if we followed it more strictly. Scrum definitely helps you to start developing quickly in the start up phase where everything is still a bit vague, which is necessary with the deadlines imposed by the Imagine Cup.

6.4 Dealing with deadlines

During the project we had three strict deadlines, that had to be fulfilled.

- Imagine Cup Round 1: March 1st 2009 23:59GMT
- Dutch Software Development Finals: April 20th 2009
- Imagine Cup Round 2: May 20th 2009 23:59GMT

Initially we planned to have a development freeze each week before the deadline so we could thoroughly test the game and produce the required documents. For the first deadline of round one we decently managed to do this, but it proved to be much harder for the other deadlines. We believe this had two reasons, the first being that we did not manage our scrum sprints properly as mentioned in the previous section. The other is that simply too much came together too late, which was not always our fault. For example, the Cannibal Composer was delivered to us just three days before the deadline of round two, in a way that we could actually do something with it. We got a version with a terrain size of 100x100 the week before, but our test level alone already was shaped 150x150, so we had to wait for the Composer that allowed us to create infinite terrain. This meant that we had to make a large part that would actually be the gameplay in the last days, making it quite hard to test.

6.5 Internal Process

Because the competitions first deadline was on the first of March, this meant that we already had to start development of the game before the fourth quarter that is reserved for Bachelor Projects. That is why we already started the project at the end of September. At first we wanted to wait until we had found an artist, since they are usually more creative when it comes down to creating game concepts, but when December drew upon us and our search was still not successful we decided it was time to start on our own.

Because our specialized software had not been installed on the computers in Building 35, we first borrowed a PC and started working in the conference room on the 12th floor. Instead of each of us starting at home, we found it very important to program the first steps of the program together because that would be the core functionality of the game, and everybody needs to be on the same page about the structure. Programming with all of us at once on the same PC also provided a good way to discuss what would be the best option for our core algorithms and squash bugs in an early stage since three people see more then one.

During the third quarter we started working in our own room in Building 35. Since we all needed to follow our third year courses, we decided to meet in our room whenever two or more people were available at the same time. In the fourth and final quarter we started working full time and we must say it went reasonably smooth. Besides some problems of people being late, we usually got our work done in time and the morale was high. The image of being able to win a trip to Egypt is certainly a big stimulus!

6.6 Testing

Most software is poorly tested, and we are sad to say our project is no different. Although we discussed in our Orientation Report we were going to use spikes, meaning we would create small worlds and demos to test each unique part before unleashing it in our large world. In practice this would mean that we would use a specific level where we could try out some things without disturbing our main level. So instead of creating new and separate tests for each component, we would use our test level and play with it in there.

Testing proved to be hard, because how do you test in a dynamic environment? One time it works and the other time it would not. So we basically played trough each component quite a few times and said it was OK when the error stopped occurring. Although this would not mean that the component was bug free, we decided it was good enough for now.

Another problem we dealt with is that different bugs would occur on different computers. This was mainly due to the fact that each PC is different and on different performance levels the game would behave differently. For example, on our PC's at home our game ran on 500 Frames Per Second, on the PC's in our room in Building 35 at 100 and on a laptop at 25 FPS. Because we had about three performance levels ranging from very high to low, it helped us identifying new bugs. By having at least two computers on both sides of the spectrum we hoped we covered our bases on that side. For example we had a bug where the game would suffer from jitter on average computers and whilst that fix worked on those computers, it created a huge FPS drop on the high-end PCs. So after a bit of fiddling we found a different solution.

In the Plan of Approach we also mentioned that we wanted to do play tests on different elementary schools in Delft. Unfortunately, because everything came together so late, we did not manage to do that. To fill that gap in some way we regularly got some feedback from the people at Cannibal Game Studios and they were able to judge with a fresh and objective look on what felt right or not.

Chapter 7

Structure

To develop a game of more than 13.800 lines of code it is very important to have a solid structure using the strength of object oriented programming. In this chapter the concepts behind the structure of the game, and the hierarchy of the different types of objects are discussed. Also the roles of the three managers: the interface, sound and time manager will be explained. The basic structure of the game is shown in figure 7.1. This is an extremely trimmed down version of the full Class Diagram.

7.1 Game Content & Logic

Games consist of various types of content combined with game logic. For example the visible shape and appearance of objects in a game are content. Also the conversation dialogs, menus and background music are content. Even positioning data, like the position of objects in a level, is an important part of the game content. All this content is structured by type, for instance: model, texture, sound, etc. The content is accessible via corresponding static classes nested in the `Assets` class, which is also static so any class can access them.

The behaviour of visible, and sometimes invisible, objects is determined by the game logic. For example the movement of a door. But game logic also determines how everyting works: from the menus, to an object you can interact with.

So almost every part of the game combines a bit of content with a bit of logic. This is why every object is a class, where the content and logic comes together in the code.

7.2 Namespaces

To organize all the code for the game logic a clean and manageble structure is required. We used the namespaces below to classify all the code and provide context for the classes they hold. Within the namespaces we used folders to group the same type of classes. Some classes are not placed in one of these namespaces below:

- `Program`,
- `Game`,
- `Assets`,
- `Settings`,
- `Player`,
- `TimeGadget`.

Although technically `Player` is a `DynamicObject` it is not placed in the Objects namespace because it has the nested static classes `Inventory` and `Progress`.

7.2.1 Cameras

The Cameras namespace contains all the different types of cameras the game uses. They are used by the interface manager to show the graphical interface of the game, for instance the main menu. This namespace is also used by all level classes, because there is a camera flying through a level to view the player.

7.2.2 GameStates

This namespace contains all the different game state classes. Since the game can be in a lot of different and complex states this namespace could potentially explode in size and complexity, but because of a clever design stayed very slim. The next chapter will go into the details of that design. The GameStates namespace is used by the `Game` class, that holds the current gamestate, and the Interface namespace for the menu actions.

7.2.3 Interface

The Interface namespace contains the interface manager, the HUD, loading screen, all the menus and all different types of menu controls. The `InterfaceManager` is a static class that keeps track of all the cameras and updates (fading) graphical user interface elements such as (parts of) menus. A menu consists of at least one selectable menu control and every menu control can have its own event handlers to handle user input. More about user input handling in chapter 8. When ingame the interface manager also displays the HUD: the energy meter and the seeds counter.

7.2.4 Levels

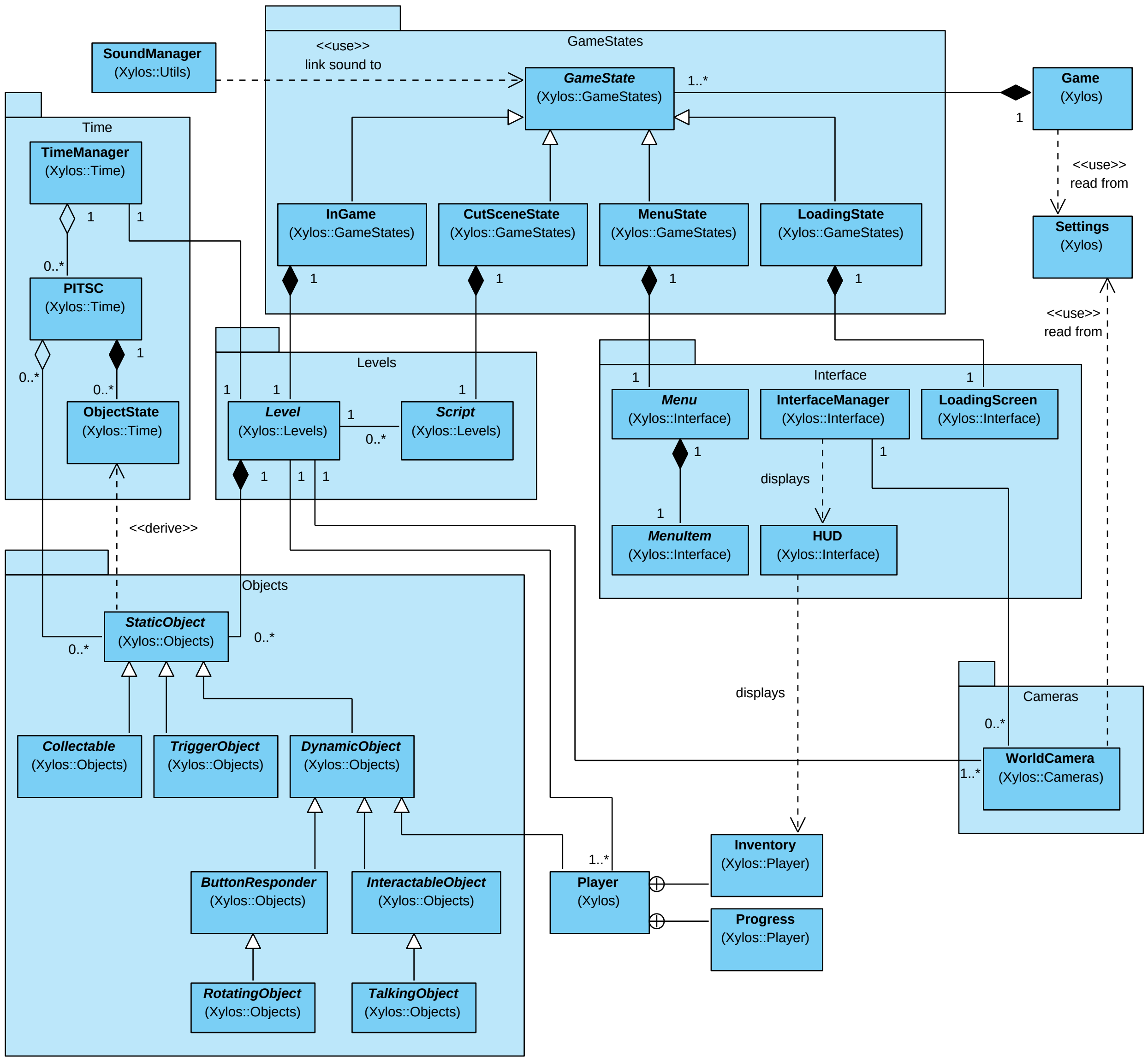
Like the name says this namespace contains all the levels of the game. Also the cut scenes are placed here, because if a cut scene would take place in the past or future the world will look different and thus a different level is needed. A level is a world in which objects and a player with a following camera are placed. Therefore all levels use the namespaces Objects and Cameras.

7.2.5 Objects

The namespace Objects is the largest namespace of the game. It contains all the visible and some invisible objects placed in a level. For example a house or an invisible trigger that starts a cut scene. The large variety of objects requires a secondary structure that may become rather complex, because there are objects that:

- can move in every direction,
- can rotatate around every axis,
- can talk,
- the player can interact with,
- all of the above, even at the same time.

But there are also a lot of objects that are passive and do nothing but prevent the player from walking through them, like a solid wall. This is why all objects can be classified as either a `StaticObject`, which is passive, or a `DynamicObject`, which is updated every frame. The complete objects hierarchy is shown in figure 7.2.



SoundManager (Xylos::Utils)
<<use>> link sound to
GameStates
GameState (Xylos::GameStates)
1..*
Game (Xylos)
1
<<use>> read from
Settings (Xylos)
<<use>> read from
InGame (Xylos::GameStates)
CutSceneState (Xylos::GameStates)
MenuState (Xylos::GameStates)
LoadingState (Xylos::GameStates)
Time
TimeManager (Xylos::Time)
PITSC (Xylos::Time)
ObjectState (Xylos::Time)
0..*
<<derive>>
Levels
Level (Xylos::Levels)
Script (Xylos::Levels)
Interface
Menu (Xylos::Interface)
InterfaceManager (Xylos::Interface)
LoadingScreen (Xylos::Interface)
MenuItem (Xylos::Interface)
HUD (Xylos::Interface)
displays
Objects
StaticObject (Xylos::Objects)
Collectable (Xylos::Objects)
TriggerObject (Xylos::Objects)
DynamicObject (Xylos::Objects)
ButtonResponder (Xylos::Objects)
InteractableObject (Xylos::Objects)
RotatingObject (Xylos::Objects)
TalkingObject (Xylos::Objects)
Cameras
WorldCamera (Xylos::Cameras)
Player (Xylos)
Inventory (Xylos::Player)
Progress (Xylos::Player)

Figure 7.1: Basic structure of the game

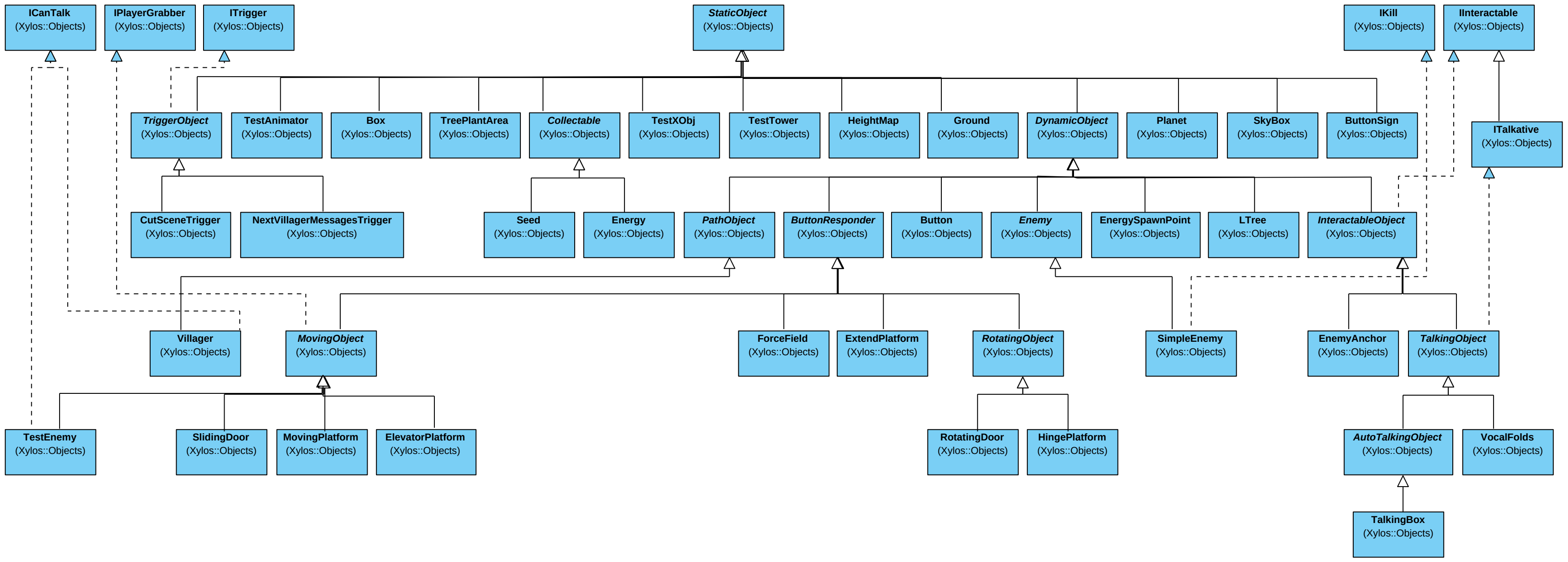

ICanTalk (Xylos::Objects)
IPlayerGrabber (Xylos::Objects)
ITrigger (Xylos::Objects)
StaticObject (Xylos::Objects)
IKill (Xylos::Objects)
IInteractable (Xylos::Objects)
TriggerObject (Xylos::Objects)
TestAnimator (Xylos::Objects)
Box (Xylos::Objects)
TreePlantArea (Xylos::Objects)
Collectable (Xylos::Objects)
TestXObj (Xylos::Objects)
TestTower (Xylos::Objects)
HeightMap (Xylos::Objects)
Ground (Xylos::Objects)
DynamicObject (Xylos::Objects)
Planet (Xylos::Objects)
SkyBox (Xylos::Objects)
ButtonSign (Xylos::Objects)
ITalkative (Xylos::Objects)
CutSceneTrigger (Xylos::Objects)
NextVillagerMessagesTrigger (Xylos::Objects)
Seed (Xylos::Objects)
Energy (Xylos::Objects)
PathObject (Xylos::Objects)
ButtonResponder (Xylos::Objects)
Button (Xylos::Objects)
Enemy (Xylos::Objects)
EnergySpawnPoint (Xylos::Objects)
LTree (Xylos::Objects)
InteractableObject (Xylos::Objects)
Villager (Xylos::Objects)
MovingObject (Xylos::Objects)
ForceField (Xylos::Objects)
ExtendPlatform (Xylos::Objects)
RotatingObject (Xylos::Objects)
SimpleEnemy (Xylos::Objects)
EnemyAnchor (Xylos::Objects)
TalkingObject (Xylos::Objects)
TestEnemy (Xylos::Objects)
SlidingDoor (Xylos::Objects)
MovingPlatform (Xylos::Objects)
ElevatorPlatform (Xylos::Objects)
RotatingDoor (Xylos::Objects)
HingePlatform (Xylos::Objects)
AutoTalkingObject (Xylos::Objects)
VocalFolds (Xylos::Objects)
TalkingBox (Xylos::Objects)

Figure 7.2: Objects hierarchy

Static Objects

From figure 7.2 it is clear that `StaticObject` is the main object that holds all basic properties and fields. It can have a collision body that defines the area where the player can not go through, but it is not the responsibility of a `StaticObject` to do collision detection. All objects, except cameras, that are placed in the world inherit from `StaticObject`. Figure 7.1 shows the three most important classes: `Collectable`, `TriggerObject` and `DynamicObject` (and their subclasses). A `Collectable` is an object that the player can collect by colliding with it and after which the collectable object disappears. And a `TriggerObject` is an object that can trigger a script or cut scene.

Dynamic Objects

A `DynamicObject` is the most versatile object in the sense that it can do everything a `StaticObject` by definition can not do. Like move, rotate, talk and interact with the player (see the enumeration above). It is the responsibility of dynamic objects to do collision detection, because they can move by themselves or because of gravity by which they are affected by default. Figure 7.1 shows the most important classes that inherit from `DynamicObject`.

The `ButtonResponder` is an object that responds if the linked buttons are pressed (or not). And since all rotating objects are incidentally also button responders the a `RotatingObject` inherits from `ButtonResponder`.

An `InteractableObject` is an object that the player can interact with, like a switch. And since talking is a form of interaction, a `TalkingObject` inherits from `InteractableObject`.

The `Player` is also a `DynamicObject` but is not placed in the Objects namespace because it has the nested static classes `Inventory` and `Progress`, as said before.

7.2.6 Time

The classes inside the `Time` namespace are responsible for everything relating to time and time travel. It contains classes that know what time it is in the world (eg: `StoryTime`), classes that store data about a `StaticObject` on a certain moment in time (eg: `ObjectState`), and classes that manage the time travel (eg. `TimeManager`. These classes will be discusses in more detail in chapter 9.

7.2.7 Utilities

The Utilities namespace is quite a large namespace, because it contains useful helper classes for various features of the game:

- Collision,
- Graphics,
- Initialization,
- Math,
- Messages,
- Particles,
- Path,
- Sound,
- Time.

The sound manager is responsible for the playback of all the sound of the game. Everything that wants to play, loop, pause or stop sound calls the, for this reason static, sound manager passing the name of the sound as an argument. It is also possible to link sound to the current game state so it will pause and resume automatically when the game state changes.

Chapter 8

Game States

In this chapter the game states are discussed, including the menus of the menu state. The game can be in many different and complex states, but always only the current game state is updated and handles all the player input. To keep control over the state of the game we defined four major game states. Since the game can be in a lot of different and complex states this may not seem enough. But by switching between these four states the game can have menus, levels, cutscenes and a loading screen. Only it won't be easy to return to the previous game state or even the one before that. Because we wanted to be able to exit a certain game state and return to it later we made put the game states on a stack. This way we can suspend and override a certain state and later resume it later. We will take a look at this approach from the perspective of the four game states.

8.1 InGame State

The `InGame` state is the most important one, because this the state the game is in while you are playing the actual game. As said above the current gamestate handles all the player input. The `InGame` state registers the player input handlers and forwards the input values to the current level. If the player pauses the game the `InGame` state is suspended and a new `MenuState` is pushed on the stack. At that point the `InGame` state will not be updated anymore until the `MenuState` is popped from the stack. This way the game pauses whenever another state is pushed on top of `InGame` state, but can just as easy resume. It is even possible to create a new `InGame` state to override another to create mini-game or sub-level.

There is a special case when the player interacts with a talking object, because at that point all user input is temporarily blocked, except a button to continue, until the conversation is finished.

8.2 CutScene State

It is possible to switch fluently from an `InGame` state to a `CutScene` state. Simply override the `InGame` state with a `CutScene` state. Just like a cut scene generally has a script that tells some sort of story, the `CutScene` state also has a script. This script can control the level of the underlying `InGame` state, and thus show everything it needs to show. This design offers a great deal of freedom. For example: if a cut scene takes place in the past or future the world will have to look different and thus a different level is needed. Just push another `InGame` state before the `CutScene` state and that is all that is needed to make it possible. During this game state the player can only pause, resume or exit the game.

8.3 Loading State

The `Loading` state is like the name suggests just a state that is active while loading a new state. In fact it shows a loading screen with a loading bar and a picture of game controls while loading the level of a new `InGame` state. The constructor of the `Loading` state calls a static Load method of the `Level` class that returns a list of content items in that level that can be preloaded. After the preloading is complete the `Loading` state automaticaly exits and change it to an `InGame` state. During this game state all player input is ignored.

8.4 Menu State

When the game shows a menu it is logically in the `Menu` state. Because `Menu` states are stackable like all other game states it is very easy to create a nested menu structure. As shown in figure 7.1 every `Menu` state has one menu that consists of at least one menu item. A method of the `Menu` state can be linked to a menu item event handler, for example a menu item "Resume game" is linked to the `exitMenuState` method which will exit the `Menu` state so the game resumes the underlying `InGame` state.

Chapter 9

Time Travel

This chapter discusses the implementation of time travel, and most of the hurdles encountered along the way. The first two sections, 9.1 and 9.2, discuss some aspects we had to look at before we could even attempt to implement time travel. The four sections that follow, 9.3 to 9.6, discuss the actual implementation of time travel.

At a cursory glance, the titles of sections 9.3 to 9.6 might not make sense. These terms come from the book 'Imagining the Tenth Dimension' by Rob Bryanton [2]. We did not implement time travel with this book in mind, but it fit rather well. Appendix B describes what these terms mean for the game.

9.1 Storage Requirements

Implementing time travel is by no means a trivial task. It is obvious that something should be recorded, but what exactly, and possibly more importantly: when? In the design phase of the implementation of this feature, we believed that the easiest way would be to save the data of every object close to the player, every frame. However, it was not clear if the amount of memory available would support this amount of data to be stored. To find out if our approach would work, we constructed table 9.1 to determine how much space would be required to store the timeline.

data type	**format**	**size** (in Bytes)
Object reference	`int`	4
Position	`Vector3`	12
Rotation	`Quaternion`	16
Misc. Data	`int`	4
Total:		36

Table 9.1: Expected sizes of saved data per object, per frame

Thus, if our game would run at 30 fps[1], we would need $36 * 30 = 1080$ Bytes per second. If we then assume that the amount of objects to be recorded at one moment will not exceed 100, the amount of memory necessary would be a maximum of 106 kBytes per second, or 6.2 MBytes per minute. Even if we include the timestamp in this calculation (which has a size of 4 Bytes), the total would still be a maximum of 6.2 Bytes per minute, since it only has to be stored once for every set of maximal 100 objects.

There is also another factor influencing the choice of how far back in time the player should be able to travel. Namely, how far does someone *want* to go back? To keep gameplay interesting, it should not be required of the player to rewind a long time, or failing to complete a puzzle multiple times will result in major frustration. Therefore, we initially kept the limit at 1 minute, which would provide the player with

[1]30 fps is the approximate lower bound where the game still seems to go smoothly

enough time to take small breaks within the puzzles. In the end, this proved to be a bit too short, so we increased the limit to 5 minutes, making the approximate upper bound 32 MBytes. It should not pose a problem to have this amount of data stored in memory, as the RAM size of an Xbox 360 is 512 MB, and that of a contemporary computer at least the same.

9.2 Time Requirements

To store the data of the objects, we would need a timestamp so we can properly retrieve them later. After a general inspection of the `System.DateTime` class, it seemed this would be sufficient for our needs. However, at that time we still had in mind the possibility of porting our game to the Xbox 360 as well. Unfortunately, the `System.DateTime` class for the Xbox 360 did not have enough functionality to be of use for our game. Therefore, we made our own version of it, making sure we could use it on both Windows machines and Xbox 360 consoles.

StoryDateTime The `StoryDateTime` class is our answer to the various versions of the `System.DateTime` class. It provided largely the same functionality as its inspiration, but is somewhat less precise. Where in the `DateTime` class the nanosecond is the smallest unit, in `StoryDateTime` the millisecond is the smallest unit. A couple of factors contributed to this choice. For one, the frame rate would have to be larger that 1000 frames per second to get into the nanosecond range (and only empty scenes render that fast, even on high-end computers). Secondly, the precision of the timestep between each update is not high enough to give meaningful values for the nanosecond range.

Besides our own `StoryDateTime`, we would also need a wrapper around it. This wrapper would keep track of the various times running through the game, and update the currently active one each frame. It should also also allow time to flow backwards, and at multiple speeds.

StoryTime Eventually, that wrapper became known as `StoryTime`. Although this class does not provide every functionality as described above, it does its job admirably. Because we never got to implement the part of the game in the era before Worf took over, only one `StoryDateTime` would be necessary to keep track of what time it is. Therefore `StoryTime` only has one reference to a `StoryDateTime`, but its other functionalities remain intact. A simple `float` value provides the multiplier for the speed of time (a negative value means time is flowing backward), which is internally accessible and settable. A feature it got later on is that it also regulates the effect you see on the screen if you rewind time. That feature was added here because this was the class every other class used to rewind time, and every necessary class to regulate the effect was accessible from here.

9.3 A Point

Because we wanted to group the saved data of objects in a certain moment of time together, we needed another wrapper. This wrapper would contain links between objects and their respective data, and also the timestamp. Eventually, this wrapper got called a `PITSC`[2]. Besides the initially required features, the `PITSC` class also queries for the data it needs from the `Level` the objects are in. That way creating and filling a new `PITSC` would be as easy as passing along the `Level` to its constructor, and it also takes away some responsibility from the classes that would use the `PITSC`.

The actual data stored in a `PITSC` differs a bit from what we thought we would store. We use a class called `ObjectState`, and its many subclasses, to store data from one object, instead of a generic structure for all types of objects. This way we do not have to store, say, rotational data, for objects that do not move at all. The `ObjectState` class only has a `long` field, although this could easily have been a `short` in

[2]PITSC: 'Point-In-Time Space-Container'

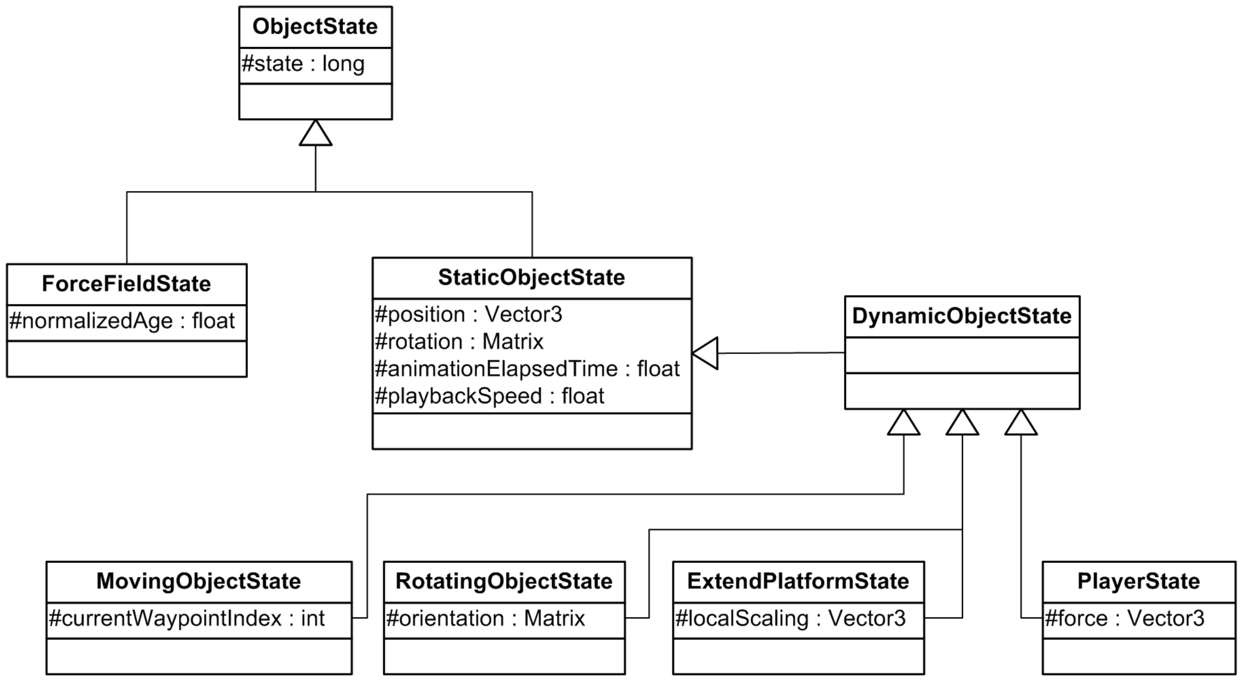


Figure 9.1: The hierarchy of the `ObjectState` class

hindsight since it is only used to store a couple of booleans and sometimes a Byte. We could have gotten rid of the `long` value altogether, but since the size of a `bool` is still a Byte, we decided to keep it. We never came round to changing the `long` to a smaller type, as the amount of memory saved by doing so would be negligible.

An overview of the `ObjectState` hierarchy can be seen in figure 9.1. With just the information in that hierarchy, some subclasses, like `DynamicObjectState`, seem redundant. However, the diagram only shows the attributes of each class, and classes like `DynamicObjectState` have more or more specialized methods than their base classes.

Another design choice worth noting is the use of a `ForceFieldState`, of which the only difference with `ObjectState` is the extra `float` field. We did not use the state inside `ObjectState` for this, since the roundabout way of storing and retrieving a `float` into and from a `long` proved to be to much of a hassle. And since the extra space required would not be very large, we decided to simply save the value as a `float` instead.

9.4 A Line

These points in time need to be stored in a way that makes browsing through them as easy as possible. For this timeline, it should be possible to have some sort of pointer to one of its entries. Also, it should be easy to move the pointer one step forward or backward. This led us to use a `LinkedList` of `PITSC`s as the timeline, using a `LinkedListNode` as the pointer to ‘now’.

This timeline is stored by the `TimeManager`, which regulates everything connected to this timeline. It also manages a couple of events related to the timeline. Because it might be awkward trying to imagine when these events take place, the two illustrations in figure 9.2 are provided to give visual examples.

① **RewindStartEvent** This event indicates time has started rewinding. It will be invoked in the first `Update`-loop the `TimeManager` notices the time multiplier of `StoryTime` flipped to a negative value.

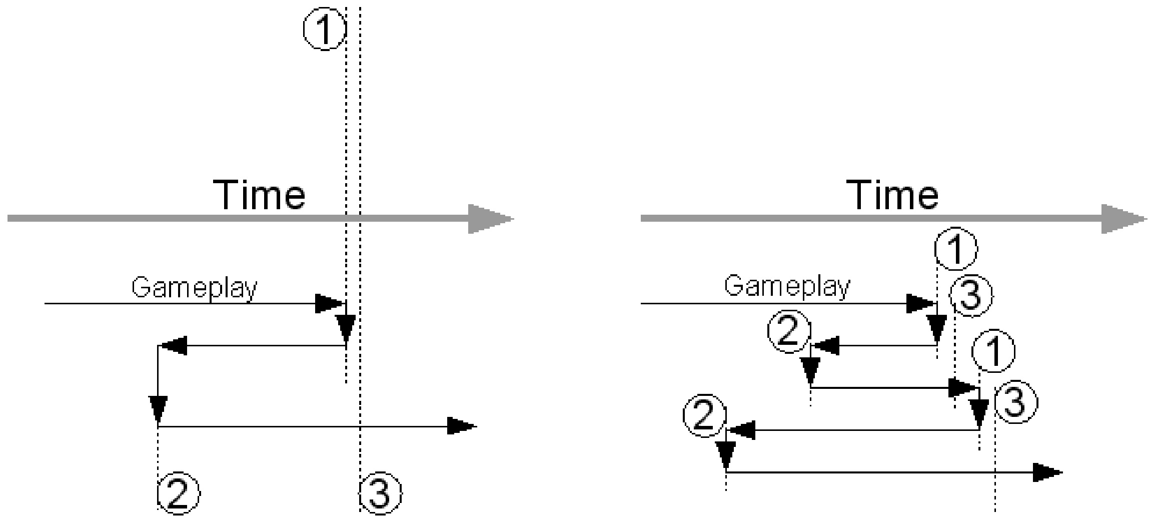


Figure 9.2: Two examples of when the timeline events are fired.

② **RewindStopEvent** This event is similar to the RewindStartEvent, but indicates time has stopped rewinding instead. It will be invoked in the first `Update`-loop the `TimeManager` notices the time multiplier of `StoryTime` flipped to a positive value.

③ **EndOfPlaybackEvent** This event is somewhat more interesting from a technical point of view. It indicates that time has come to a point where it has not been yet, while it was still at a point it had already been in the previous frame.

Filling the timeline

When filling the timeline, one should be careful when to query for data. It should either happen before any object in the world is updated, or after everything has updated. When this happens initially seemed only like a matter of choice, although we should keep in mind that choice when designing and implementing other aspects of time travel. We chose to save the data at the start of each frame.

As was mentioned earlier, saving the current state of the objects in the world was relatively easy. Adding a new `PITSC` to the end of the timeline would be enough. In the constructor of the `PITSC` the `Level` would be queried for all `StaticObjects`, from which the state was determined. Of course, the `PITSC` did not determine the state of the objects, but asked the objects what state they had.

9.5 A Split

We have chosen the data in the `ObjectStates` in such a way that, using this data, the original object can reconstruct itself to the state it was in when the `ObjectState` was taken. Or, in other words, an `ObjectState` is a snapshot of an object, which the object can reconstruct again. Thus a `PITSC` is a snapshot of the changing objects in the world, which can be reconstructed.

This means rewinding time can be done fluently, if only the process of reconstructing does not take too much time. However, the only way to find out if it will run at decent speeds is to implement it. After we did, only a slight but acceptable drop in the frame rate was measured.

To implement this, some hurdles had to be overcome, however. The first and major one was the fact that objects could (theoretically) appear into and disappear from the world when time flowed forward. This meant we suddenly did not have enough information to reconstruct a `PITSC` if we only used one of them as reference. Therefore the method that reconstructs the `PITSC` also requires the 'previous' `PITSC` in the timeline to be passed on as an argument[3].

Another hurdle was more of an error on our side, which happened once a `PITSC` is reconstructed for a second time. Before the bug can be explained, some more implementation details should be given, however. If an object is asked to reconstruct one of its previous `ObjectState`s, it first applies the data it added itself, and then passes the state on to the same method in the base class. However, for the base class to be able to properly interpret the `ObjectState.state` field, any other data added to that field by subclasses should be removed, so we did. However, we did not put the data back again after the base class applied the rest of the `ObjectState`. And since `ObjectState`s are passed on by-reference, the data was removed from the `PITSC`, and could not be retrieved the next time the object reconstructed the data from the `PITSC`. This matter was easily resolved once we found out what was wrong.

9.6 A Fold

The final step in implementing time travel is replaying the timeline, while being able to change certain things, and still keep the timeline consistent. On top of that, some things should *not* be allowed to change, and the previous versions of the player should be visible as well. Both the ability to change certain things and the inability to change other things, and the consistent timeline were the source of some frustration during implementation. However, they were also the most interesting to implement properly.

9.6.1 Volatility

As mentioned in section 4.2, we designed the time traveling gameplay to be as realistic as possible. We also designed time travel in the game to prevent as much paradoxes as possible, and to ignore the ones that still slip through. This means that objects you interacted with (and thus the previous version of yourself *will* interact with) should not be allowed to alter from their recorded path, as this could create a paradox. An example would be using a platform to get to an energy spawn point (you had an empty energy meter), then rewind, and move the platform using another button *before the previous version could use the platform.* Now the previous version of yourself should not be able to reach the energy spawn point, preventing him from going back in time. However, the current version of yourself *did* go back in time, so the previous version of yourself *should have* been able to reach the energy spawn point, resulting in a paradox.

The objects you interact with and include, but are not limited to: seeds, buttons and platforms. Each of these objects required a slightly different approach. The general approach however, was the use of two boolean flags; `IsAlteredByPlayer` and `CanBeAlteredByPlayer`, which are stored as fields in every `StaticObject`. The first indicates the object has been changed by the current version of the player; the one you are now controlling. The latter indicates the object has been changed by a previous version of the player, and thus should not be altered by the current version of the player. When `IsAlteredByPlayer` is true when rewinding starts, both that field and `CanBeAlteredByPlayer` are set to false, rendering the object unalterable by the player. This is done via a method in `Level`, which is hooked to the `RewindStartEvent` from the `TimeManager`. Figure 9.3 is a sequence diagram[4] describing how these are used to render a Seed unalterable by the current version of the player. Similarly, every object locked by `CanBeAlteredByPlayer` will be unlocked once the `EndOfPlaybackEvent` fires.

[3]While rewinding, the 'previous' `PITSC` is the one with the slightly higher timestamp. If time is moving forward, it is the one with the slightly lower timestamp

[4]Note that while the image might not be a proper UML Sequence Diagram, it should still convey its message.

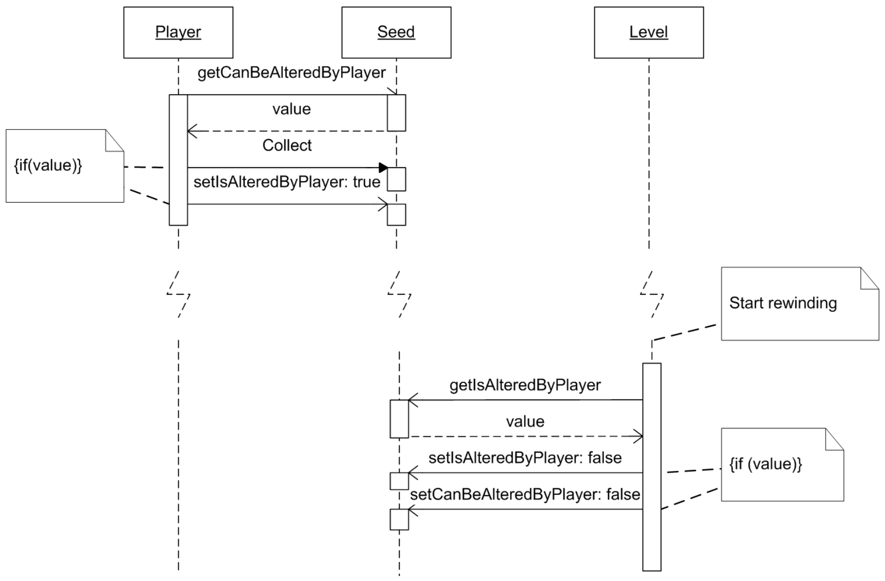


Figure 9.3: The sequence of events leading to a Seed that cannot be picked up a second time.

Seeds

Seeds are special in the way that they disappear from the world if you interact with them. Setting the `IsAlteredByPlayer` flag will then have little effect, because the query looks for `StaticObjects` *within* the world. We solved this by incorporating Snippet 9.1 in the code that loads a `PITSC` into the world. This check is performed for each `StaticObject o` that is in the `PITSC` to reconstruct, but was not in the 'previous' `PITSC`. The code basically does the same as the method in `Level` that hooked on to the `RewindStartEvent`, but for objects that spawn because of the replaying of the timeline. However, it also reduces the opacity of the object, making it somewhat transparent.

```
/*
 * if we're going back in time, and an object that the player has altered is going to spwan again,
 * we should set the values for IsAlteredByPlayer and CanBeAlteredbyPlayer to false.
 * this should prevent the object from being used before a previous version used it.
 */
if (o.IsAlteredByPlayer && TimeManager.TimeFlow == FlowOfTime.BACKWARDS)
{
    o.IsAlteredByPlayer = false;
    o.CanBeAlteredByPlayer = false;
    level.Add(o);
    o.SetOpacity(Settings.PastTransparency);
}
```

Snippet 9.1: The code that makes seeds (and other collectables) impossible to pick up a second time.

Buttons

Buttons must respond to any player that stands on it, including previous versions and current versions. Initially, we treated buttons like every other object: once you interact with it (by standing on it) and rewind, you cannot change the button state until the previous version of the player is gone. However, when testing this mechanic, this proved to be highly annoying. If a small mistake had been made, and you accidentally also pressed the button for a very short while before you reversed time, you still had to wait until the `EndOfPlaybackEvent` before you could press the button again.

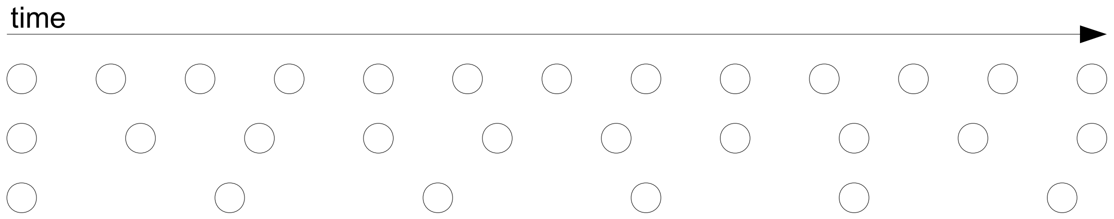


Figure 9.4: How the consistency problem manifests itself. Each circle represents a point in time where a PITSC is supposed to be saved. The top line of circles is when the player has not yet gone back in time. The second row is when there are only a few previous versions of the player in the world at the same time. The bottom row is when there are a lot of previous versions of the player in the world at the same time.

Thus the button mechanic was changed so that `IsAlteredByPlayer` is never set for buttons. Instead, it will respond to any player in range, be it a previous version or the current version. We were aware of the possible paradoxes that could occur because of this, but decided to prioritize gameplay over the prevention of possible paradoxes.

Platforms

Platforms are a bit of a mix between the method used for seeds and for buttons. They do use the `IsAlteredByPlayer` flag, but since they respond to buttons being pressed or released, they are still allowed to deviate from their recorded states.

9.6.2 Consistency

The problem of consistency proved to be a problem harder to tackle. It would not have been a problem if there were no drops in the framerate when replaying part of the timeline. Part of the problem is illustrated in figure 9.4. We did not realize this was happening at first, and sometimes saw objects that wanted to be at two different places at the same time. We rewrote the code that creates the `PITSC`s, keeping this this problem in mind. Instead of updating the `PITSC` that was loaded at the start of the frame, we would now create a new `PITSC` at the moment in time the frame is rendered. Also, we delete any `PITSC` between the one just saved and the one saved in the previous frame, making the timeline consistent again.

Another part was easily resolved, and involved us making the wrong choice because our assumptions were false. As mentioned in section 9.4, we chose to save the data of the world at the start of each frame. However, when replaying (part of) the timeline, each frame a `PITSC` has to be reconstructed as well. In order for the player to be able to interact with the new state of the objects, the `PITSC` has to be reconstructed at the start of the frame. If we did not move the saving of the new `PITSC` to the end of the frame, the timeline would not have been updated, except for the state of the current version of the player. We could possibly have moved the saving even further to the start of the frame, even before the `PITSC` was loaded, but since moving it to the end seemed to work, we did not bother to test alternatives.

Chapter 10

Trees

Early on in the design we wanted some form of procedurally generated content in our game, if only because the team only consisted of three programmers back then. Our thoughts quickly settled on having the trees in the game world generated procedurally. One of the reasons for this was that we did not want every tree to look the same, without having to sculpt an insane amount of them. After extensive research in the field of procedural content generation for another course [1], it became apparent that using a method called L-systems would quickly produce good-looking trees. As an introduction, L-systems are described in section 10.1

The idea got reinforced when the decision came that the main character was to plant seeds that grew into trees quickly. Using procedurally generated trees, showing the animation of a tree growing was going to be as simple as drawing the intermittent stages of the generation of the tree. The actual implementation of the growing trees is described in section 10.2

Because of the nature of our most prominent game mechanic, time travel, the trees also had to be able to shrink again. This seemed easy at first, but we encountered quite some performance problems using the quick and dirty method. Section 10.3 describes how we have integrated time travel in the trees and the underlying L-systems.

10.1 L-systems

The basics

L-systems are based on the mathematical concept of formal grammars, and some are equivalent to a formal grammar. An L-system is an iterative structure, which is defined by an initial string, two sets of characters defining the alphabet and the constants, and a set of production rules. A production rule contains at least two elements: the *antecedent* and the *consequent*. The *antecedent* can only contain characters from the alphabet, the *consequent* can contain characters form both the alphabet and the set of constants.

To advance the L-system with one step, the production rules need to be applied to the current string. In contrast to formal grammars, where only one production rule is applied each iteration, as many production rules as possible must be applied for L-systems. To apply a production rule, replace an occurrence (or all occurrences) of the *antecedent* in the current string with the *consequent*.

Introducing randomness

Given the same input string, one of these basic L-systems will always generate the same set of strings, no matter how many times you run it. Therefore, to use L-systems to generate more interesting procedural content, some form of randomness should be introduced. There are multiple mays in which randomness could be introduced, some of them are described below.

Randomly keep a rule from firing One could choose not to fire one or more rules at random. Depending on the set of production rules present in the L-system, this approach can be both helpful and almost not helpful at all in introducing randomness. If none of the rules have common characters in their antecedents, this approach will have little effect. All it would accomplish is that the temporarily disabled rule will still fire on the same character(s) in one of the next iterations. However, if there are a lot of antecedents sharing the same characters, this approach should produce some decent results. When a rule cannot fire, the characters it would have used become available for another rule to be fired, preventing the disabled rule from firing the next iteration instead.

Introduce alternative consequents Another option would be to introduce alternate consequents for most rules. Then, when a rule fires, it is up to chance which consequent is used. If the consequents are chosen diverse enough, this method can be very effective in introducing randomness. One could also add a probability to each consequent to tweak the resulting output, if the equal chances do not provide enough diversity among the results.

10.2 Implementation of the trees

The generic `LSystem`

Because we might want to use L-system later on as well, we decided to implement a generic L-system, in a class wittily called `LSystem`. From the start we included the option for a rule to have multiple consequents. We chose for this method since our trees would not need a lot of rules, and thus randomly disabling a rule would not produce the desired effect.

Initially, when constructing an `LSystem` object, one would have to pass on the four parts defining the L-system. Thus, the alphabet, the set of constants, the initial string and the set of rules had to be known beforehand. This actually never changed, but later the option was added to add more rules to the L-system. We decided to do this mainly because the format of the set of rules was not very elegant, as can be seen in Snippet 10.1. Using a separate method to add a rule, or an extra consequent to a rule, with its respective probability would make reading the code much easier.

```
/// <summary>
/// The set of rules of this LSystem
/// </summary>
protected Dictionary<string, List<Pair<string, float>>> rules;
```

Snippet 10.1: The definition of the set of rules in `LSystem`

Our method that advances the L-system to the next step of the iteration, does not exactly work as the definition on an L-system, as described above, implies it should. We decided not to implement a method that would maximize the amount of rules to be fired each iteration. Because of the simplicity of the L-systems we were going to use, a greedy approximation to the optimum solution will oftentimes provide the optimum solution itself. Our algorithm searches the current value string, from start to end, for substrings a rule matches on. When a rule is found that could be applied, it is applied and the search continues, until all substrings have been searched. It should be noted that, of course, characters cannot be used for the application of a rule twice, and the algorithm makes sure they do not.

Optimizing the `LSystem`

To the observant eye, it would be apparent that the algorithm described above runs on $O(n^2)$ time, and can be heavily optimized. We were able to reduce it to $O(n)$, but only with another restriction added to the `LSystem`, namely that it had to be a *Context-Free* L-system. A *Context-Free* L-system is an L-system where the antecedents of the production rules all consist of only one character. Therefore, to advance the

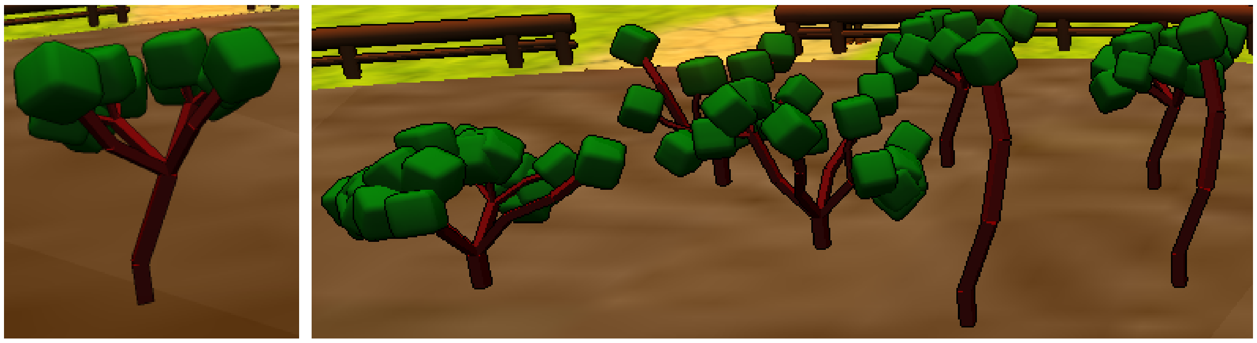


Figure 10.1: Some examples of the trees our `Turtle` can produce.

L-system, only one check per character in the current value is needed to determine the value in the next step of the iteration.

However, this would greatly reduce the amount of L-systems we could use. Luckily, we are still talking about very simple and abstract trees, for which Context-Free L-systems would be more than enough.

From string to tree

Now that we have an L-system in working order, we need to draw the strings it produces in the 3D game world. A technique often used to do this is called *Turtle Graphics*. With this technique, the strings generated by the L-system are strings of control characters ('fodder') for a 'Turtle' that can draw 3D objects. The Turtle is fed the characters of the string one by one, and moves, rotates, and places models in the 3D world, depending on which characters it is fed.

Our implementation of the Turtle, contained in the properly named class `TreeTurtle`, does exactly that. If it is fed a string of the proper format, it produces trees as seen in Figure 10.1. For the implementation of the mapping between characters and actions, `delegates` are used. For each character it is fed, it looks up what delegate to call in its internal `Dictionary`, and calls that delegate.

In the earlier versions on the implementation of `TreeTurtle`, the entire Model of the tree was built using copied data from sculpted models. Because of the amount of data to be copied again and again when growing multiple trees with a lot of leaves (the rounded blocks), the frame rate would drop significantly, even on a high-end computer. After we changed the implementation so that the same Model is drawn where each leaf should be, the drops in the frame rate were gone.

The L-system we used for our trees is not very complicated, although it might seem so at first. The definition is given in Snippet 10.2. To really understand this definition, one should know the character-action mapping of the `TreeTurtle`, which can be found in Appendix A. The definition of the constants, variables and initial value should be self-explanatory, but the definition of the set of rules might still be hard to understand. In words, the set of rules is defined like this:

- Add a rule, with antecedent `L`, and two consequents with equal probabilities (0.5).
- The first consequent makes the tree branch in four directions, while making the branches a bit shorter and thinner.
- The second consequent elongates the tree with a branch at a slight angle.

```
constants:     "+-<>^v%#_~[]"
variables:     "FIL"
initial value: "FL"
rules:         "L" -> {("_#[<<FL][>>FL][^^FL][vvFL]%~", 0.5),
                       (">F++<vL", 0.5)}
```

Snippet 10.2: L-system definition of a tree

10.3 Integrating time travel in trees

Because of the nature of L-systems, it is generally impossible to revert the process of advancing to the next step of the iteration. However, we had to be able to because time could be reversed, and it would be silly if a tree you just planted did not shrink again. We first tackled the problem of implementing advancing a negative amount of steps in `LSystem`. The solution proved not to be very difficult, but we did not get it on our first try. We first tried to rebuild the system from the ground up, using the initial value we saved. We kept in mind that the tree should look the same as it did before, so we also saved the initial random seed of the pseudo-random generator. However, this let yet again to quite a performance drop when the amount of trees became too high, but still in the range of trees the player could plant at one time.

In the end, we made a cache with the values of the previous iterations. Each time the L-system advances to a new iteration, the value is stored in the cache. If the L-system advances to an iteration it has already been in once, the value from the cache is used instead of calculating a new one. This also prevents the system from obtaining inconsistent values as it iterates once again in the 'correct' direction (into the future), without the need to store the random seed.

A similar approach was used for the trees themselves. A cache with models was made, which was used when going back in time, or when the tree came into an iteration it had already been in. The underlying `LSystem` updated simultaneously, but it is not consulted if the model can be obtained form the cache.

Chapter 11

Use of the Cannibal Composer^{TM}

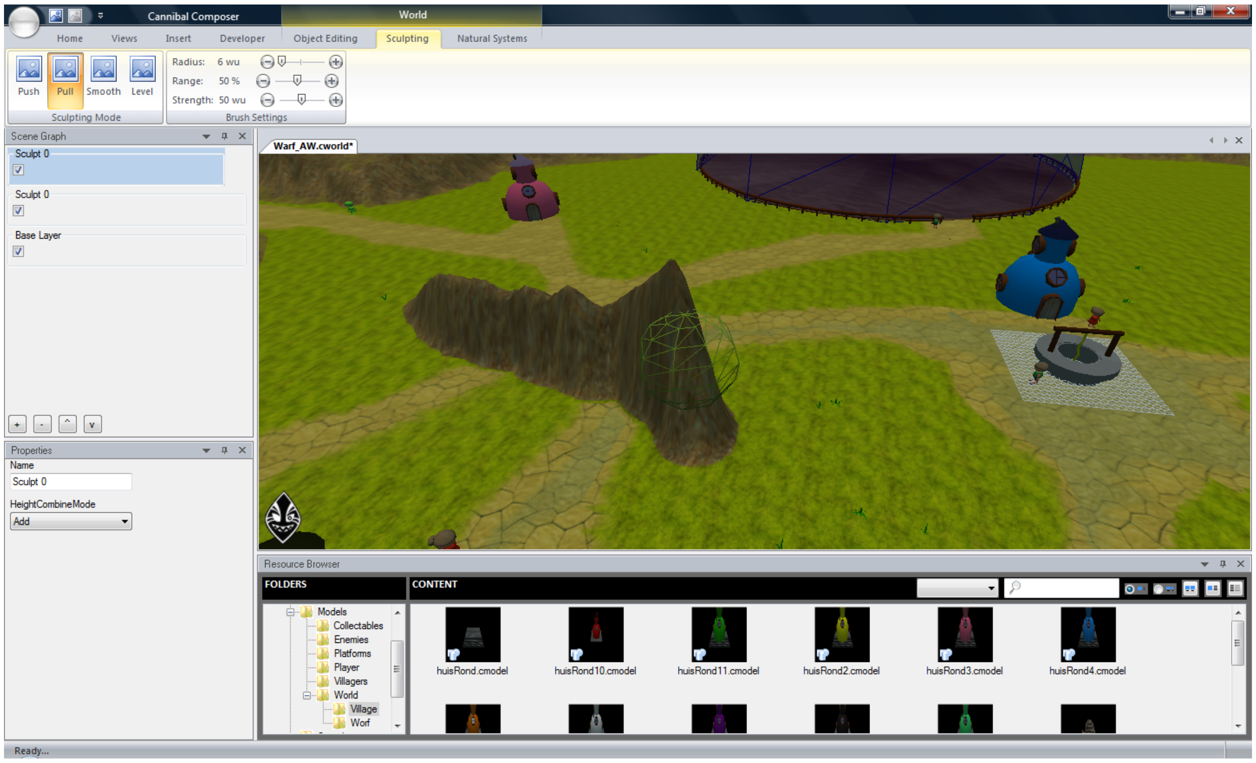


Figure 11.1: The Cannibal Composer

The Cannibal Composer is a tool created by Cannibal Game Studios that allows you to create a game world in a convenient way. Whilst normally we would have needed to put the locations of objects hardcoded in a world, the composer allowed us to simply drag and drop models in it. This saved us a lot of time because otherwise our process would be: 'Change a digit - Compile - Check if it looks ok - Change the digit into something which *might* look better'. Besides dragging simple models in the game, the composer also allows you to create ComposerObjects. A ComposerObject is a model, or collection of models, with added collision bodies. This allowed us to rig a model in a convenient way with collision bodies and rapidly import it in the game.

The composer also makes it possible to sculpt a the heightmap so you can easily add hills to your landscape to make it more alive. Finally, there are ways to edit properties of objects like DiffuseColors of objects and transparency.

11.1 Composer Converter

As you may have read in the previous sections, the Cannibal Composer came quite late. At the beginning of the project, when we did not know the specifications of the Composer, we already had created an inheritance structure. We wanted every object to be either a Static- or DynamicObject and thus we would have to convert every object to behave like this. Another problem we had, was that the ComposerObject only took an empty constructor. In the structure we created before we had the composer, we decided that for example an energy collectible could have an argument saying how high it's boost level should be. So how would we be able to assign a value of the boost? The third and final problem we had was that the ComposerObject would import the Childnodes as protected Childnodes, meaning we could not assign certain variables of those Childnodes.

To circumvent these problems we came up with the Composer Converter. The way the composer works is that it creates a `cWorld` which contains all the information about the nodes contained in it. The `ComposerConverter` then opens this world and adds all the nodes contained in it to a `List`. We then start looking at the names of each node to see of which type they are before adding them to a `Level`.

The Converter allows us to transform the objects generated by the Composer into objects we created at the start of the project. The way we achieved this was by naming all the objects in the Composer with an unique format created by us. For example `energyspawn_50_ontopofthehill` would mean the object is a energyspawn, with boost 50, and an (optional) name to keep it organized for us.

```
private static void AddEnergySpawn(Level l, Node n)
{
    //energyspawn: + boost (float) - name
    AssertPrefix(n.Name, "energyspawn", "EnergySpawnPoint");

    string[] parms = GetParameters(n);
    Assert(parms.Length >= 1 && !string.IsNullOrEmpty(parms[0]), "EnergySpawnPoint-nodes have a required
        1st parameter: boost (float)");

    EnergySpawnPoint e = new EnergySpawnPoint(n.Position, n.RotationMatrix, ParseHelper.parseFloat(parms
        [0]));
    e.Name = ConcatenateParamsAfter(parms, 1);

    l.Add(e);
}
```

Snippet 11.1: An example of converting an `EnergySpawn ComposerObject` into an `EnergySpawnPoint`

So basically the most information about an object created by the composer comes from the name assigned to it. We only use the object itself to get its position and rotation matrix, so we would know where it is located in the world and how it is oriented.

We also used the ComposerConverter to declare a group of objects as a puzzle. For example, if we added a button and a platform in the world, how would it know whether or not they are connected? We also did this by abusing the names in the composer. The name `elevator_start_p01` would declare the object as a starting position of an elevator platform called `p01` ('puzzle one'), whilst the middle word would be stop if it was the end position of the elevator. The same goes for the buttons connected to the elevator: `button_1_el_p01_1` would mean its a button, that is affecting *one* component (we could also let two elevators rise if the button would be pressed), connected to the `elevator` called `p01` and the button needs to *on* to move the platform.

11.2 Difficulties

Besides the composer being late and incompatible with our previous made up structure, we had different issues as well. Because the composer was not in its final state yet, it was sometimes quite buggy which caused some frustration. There was more then one occasion where the composer crashed midway, losing all the work we had achieved so far.

Another problem was that because of the unlimited terrain we were now able to create, Cannibal has changed the structure of the old `Terrain` (which only allowed you to load a HeightMap-image), forcing us to write a new collision system for the newly created `TerrainC`. Although the `TerrainC` was more advanced in size and the ways we could manipulate it, it was less advanced as the old `Terrain` in ways of getting information out of it. This hindered us to recreate our old collision system, leaving us with a less accurate one.

Part IV

Evaluation

Chapter 12

Reflection

Now that the project has come to an end, it is time to reflect on it. Of course, it is unfortunate that we did not make it into the finals, but in hindsight it was to be expected. More on that can be found in section 12.2. But before that, section 12.1 contains the reflection on the process.

12.1 Reflecting on the process

If we are honest, the process did not went as well as one could have hoped. First and foremost, we spent a lot of time looking for an artist, which we did not find until we were not looking any more. The problem here was that most game development educations deemed their pupils too inexperienced to work alone in a team of four, and the ones that were actually capable enough did not have the time to do so. We think this problem might be overcome by making the division two by two. By doing it this way the two coders can code and the artist design, model, animate etcetera. Now we had usually did the same, which sometimes meant that one coder was not coding but doing things that are usually not part of a Computer Science student its curriculum.

Second, we were working with tools that were not completed. Although the people at Cannibal Game Studios were a great help for us, it does not help to develop a game on an engine that is far from completed. It certainly would have helped us that if we wanted to create something, for example particle effects, we immediately could start on them. Instead we were waiting a few weeks until a third party has created support for them. Their composer was certainly a real help in the end to quickly create a world, but if it came sooner we would not have needed to create the world as quickly as we did.

We also did not have a clear overview of our first level and the way we were going to present it to the player. We kept on working on the technology behind it, which caused that we actually started working quite late with putting all the pieces of the puzzle together. Before combining all the elements we wanted to have all the pieces and we should have started combining them as early as possible.

Another problem we had was that we all were the boss and a boss is not going to punish himself if he misses a deadline. This means that if we missed a deadline, nothing happened and we simply continued with our task like nothing had happened. We think it is hard to avoid this trap since we are all third year students so we do not have authority over other each other. The only thing we can about this is take firm roles at the beginning of the project and make each person responsible for a task of its own.

What did go well however, was that we worked on the game whenever we had some spare time. This meant that even during the busy exam period, development of the game had not come to a halt. We took care not to fail the exams unnecessarily though, and did not press each other to complete feature X within Y time, when that would mean one of us would fail one of his exams.

12.2 Reflecting on the product

Although we like the game we had delivered, we still think we could have done a bit better on multiple levels. First of all, the quality of the game: because everything came together quite late we really needed to stress for the last deadline and this did not do the game justice. In the version we handed in for round two there were quite some bugs that did affect the gameplay. For example it would not continue to the world after the intro sequence and there were problems with some forcefields not functioning properly. Also there were some elevator platforms that did not stop at their halting positions. The reason for all these embarrassing errors is that we did not had the time to put the game trough a full play trough before we handed it in. Time was running out and it was either submit the game or miss the deadline completely with nothing handed in.

Another thing we should not have done was to incorporate the MDGs in a subtle way. From what we could see from the games that advanced to the finals in Egypt, their games were Millennium Goals with a bit of game on the side, whilst we opted for a fun game with a Millennium Development Goal sauce on top of it. To be frank, we hoped this would make us unique and stand out from the crowd, but if we wanted to advance to Egypt we should have gone with the approach that proved to be effective.

The fact that our game could not run on the Xbox 360 might have changed our chances of winning for the worse. Although this is impossible to check, the games that passed all seemed to have Xbox 360 compatibility. It has always been our wish for an Xbox 360 version of the game, but that was impossible because the Cannibal Engine currently only supports Windows games.

An aspect of the implementation that should be mentioned, is the effectiveness of the current implementation of the timeline. While it works, it takes up an unnecessary amount of data. We initially thought of saving only the performed actions by the objects, but we chose for the simpler method instead. But even this method could be optimized quite easily. In its current form, in a lot of consecutive `PITSC`s contain almost the same data. What if the `ObjectState`s inside would not store the absolute positions, but the differences instead? This would mean a restructuring of the `ObjectState` hierarchy however, as otherwise it would have little impact on the memory usage. For example, there would be an `ObjectState` subclass that only had a position-difference vector, or one that only had a byte with some changed flags. This would decrease the amount of used memory significantly.

Chapter 13

Conclusion

To conclude we want to say this has proven to be a fun, yet challenging project for us. It is great to create a game for your Bachelor Project and even have a shot at winning a trip to Egypt while doing it. Unfortunately, in the end we did not advance to the finals but it was fun believing we had a shot of winning whilst competing against other teams.

Another nice aspect was that we helped Cannibal Game Studios putting their product to the test and provided them with much needed end user feedback to improve their product. We hope our feedback will take their product to the next level and unlocks its full potential.

And although we did not manage to submit a game that met our quality standards before the second round, we did manage to deliver a fully working game that would be fun to play in the end. We achieved this by doing everything from scratch ourselves. Whilst normally you would have different departments responsible for different areas of game development, for example you would have different people or devisions coming up with gameplay, story design, artwork etcetera we all managed to do that on our own. We believe we succeeded in fulfilling this challenging task with the delivery of our game Xylos.

We also proved that it is possible to implement time travel into a 3D environment. We hope that some day we will see this in a commercial game as we believe it is incredible fun and provides the player with unique challenges. Another aspect of the game we are proud of is the ability to grow random generated trees so no tree has to be the same. It would have been impossible to let an artist create all those different kinds of combinations we are now able to create.

But apart from fun, exciting and interesting, the project could be hard as well. In one way spreading the Bachelor Project over a year was a good thing, it is certainly needed for a project like this, but it also could be quite stressful. For example, most of the ‘white week’ before the exams and during the exams in the second and third semester we were still working on our game, while it would possibly have been better if we could fully dedicate that time to study. But besides the exams it also meant that we had a busy second and third quarter, and finding hours where we could work together on the project was quite challenging.

All in all, we certainly think this project is worth while repeating for next year, be it that they take our points of recommendation given in the next chapter into account. We know we would do it again if we could!

Chapter 14

Recommendations and Future Work

14.1 Recommendations for the BSc Project

Creating a game, nearly from scratch, is always a fun and exciting experience. Especially if there is some carrot hung in front of you to make the game even better. That carrot is, of course, the potential of winning the *Imagine Cup*, and it is a very attractive carrot. We certainly recommend for this type of Bachelor Project to continue to exist.

However, we cannot say if it would be a wise choice to make the use of the Cannibal Engine mandatory next time. Of course, it is a great tool to create a game with, especially in combination with the Composer, but it may not be the best choice while it is still in development. We therefore believe the choice of using the engine or not should remain with the students participating in this project. But since the Cannibal Engine is no longer used in the MKT4 project, some examples or other information on what the Engine can do and plans to do might be in order, so that the students can make a better choice.

Although we know that it falls under our own responsibility as well, we think it would be good for the supervisor to hint to a possible deadline of the project's thesis as well. During development, very little documentation with respect to the Bachelor Project has been made. If we had known that we would not be able to give the presentation during the summer, we might have started on the thesis earlier. Still, it was also our own responsibility, so this might as well be a recommendation to future participants for this Bachelor Project.

14.2 Recommendations for future participants

Students wanting to participate in this Bachelor Project most likely have finished the MKT4 project as well. There, they got a game design handed on a chromed platter, which they had to make into a playable game. This project is a great opportunity to experience designing a game as well, and it made us appreciate what our colleagues from the HKU did. Although finding an experienced game designer in your team is almost a must, we highly recommend participating in creating the design. We will actually recommend to get two 'artists' on your team: one experienced game designer, and one graphical artist (be it a 2D/3D modeler, an animator/rigger, or concept artist) together with two programmers. This allows you to create most of the game, without the need to create an unofficial team of five or more (after all, only the four registered members can come to the finals for free).

We also recommend to at least give the Cannibal Engine a try, if they let you. It is already a powerful engine, and it keeps getting better and better. The combination with the Composer also makes the life of the level designer in your team easier. And do not underestimate the advantage of having a support crew a few hundred meters away (if you choose to work somewhere on the faculty, that is).

14.3 Into the future

If work on the game is going to be continued, the first thing that is going to happen is most likely the deletion of ugly last-minute hacks. These were introduced in the last few days before the final deadline, and they do not contribute to the cleanness of the code. While useful then, there are better, more structured and more efficient was to implement them.

Right after that there are some bugs that need to be fixed. Most notably the collision detection and collision response between the player (and enemies) and the terrain. A possible solution might be just some tweaking with the current algorithm, or the use of something like Navigation Meshes instead. Although implementing Navigation Meshes (and creating them) would be significantly more work, that would probably be the solution of choice.

Other than those hacks and bugs, the gameplay needs to be extended. Because we only had access to the version of the Composer we could create the world with three days before the deadline, the size of the world and puzzles has decreased significantly. It could not have been much larger, because besides sculpting the terrain all storyline events had to be placed as well, as well as the puzzle elements other than the terrain itself. And of course it had to be tested as well, which has not happened often enough in the 'final' version.

However, we have decided not to continue the game, at least not in it's current form. We have yet to decide what form, if any, it will be continued in, and who the team will consist of if it will be.

Part V

Appendices

Appendix A

TreeTurtle character mapping

- `F` → Go forward, creating a branch
- `I` → Go forward without creating a branch ('Invisible')
- `L` → Create leaf
- `^` → Pitch up
- `v` → Pitch down
- `<` → Yaw left
- `>` → Yaw right
- `+` → Roll left
- `-` → Roll right
- `#` → Thinner
- `%` → Thicker
- `_` → Shorter
- `~` → Longer
- `[` → Save orientation and push it on the stack
- `]` → Pop orientation from the stack and load it

Appendix B

‘Triads’

In his book ‘Imagining the Tenth Dimension’ [2], author Rob Bryanton explains his view of the ten dimensions and implications of this view. This is not the place to get into the book in detail, and more information about the book can be found on the website for the book[1]. On this website, an informative video can be viewed, illustrating this ten-dimensional view. To prevent a wordy reiteration of the video in this section, it is assumed that it has been viewed and understood, at least up to and including the fourth dimension.

A Line

As seen in the video, the fourth dimension is a line. And, quite literally, this is the ‘timeline’. The game is at least four-dimensional because it is a timeline of a three-dimensional world[2].

A Split

With an added fifth dimension, splits from the fourth dimension result in alternate timelines. Because the player is not limited to the same actions he did before, the game is at least six-dimensional.

A Fold

The sixth dimension is a fold, which would allow someone to instantly travel from one timeline to the next. In the game however, the player has no need to travel to and from alternate timelines. The sixth dimension manifests itself in the game by the fact that the players from other timelines are also visible, and interacting with the timeline the player is on now. Alternatively, it is manifested by merging several timelines into one.

[1]http://www.tenthdimension.com

[2]It would be at least three-dimensional if it would be a timeline of a two-dimensional world

Appendix C

Plan of Approach

November 2008

Preface

This document describes the requirements for our Bachelor of Science project *Xylos*. The goal of this project is to create an award winning game for the Microsoft Imagine Cup 2009, or at least a game that is capable of reaching the world finals. The competition is divided into three different rounds, each round we will need to submit our game and be judged by the jury. After the first round 150 teams will advance, and only six teams will advance to round three: the world finals in Egypt. The game will be made in cooperation with Cannibal Game Studios.

Summary

For this unique BSc project, the students need to start working as early as September if the final product, a game to enter in the *Microsoft Imagine Cup*, is to be of a decent quality. Design will start in September, while implementation starts in January. The game will be made in C#, and built on top of the Cannibal Engine.

The deadline of the first round is the 1st of March, in which a playable game demo must be delivered. The deadline of the second round is the 20th of May, in which a 100% playable game must be delivered. It is not required, although desirable, for the game to pass the first or second round, nor is it required to win the competition.

The development method used will be the agile method *Scrum*. Development will take place in room DW0.011 in Building 35, and until that room is prepared, a single PC has been arranged for use inside the EEMCS building. There will be regular updates from the team to the supervisor, Dr. Ir. A.R. Bidarra, and a weblog will be maintained at http://www.teamwarp.nl.

C.1 Introduction

This project is not like any of the regular BSc projects. For example, this project does not really have a sponsor, other than the sponsor of the competition, Microsoft. However, Microsoft will not be involved in any part of the process, aside from setting the deadlines. The mentioned competition is the *Imagine Cup Student Competition 2009*[1], or more specifically: the *Game Development track* of said competition. This is a worldwide competition, where students in teams of four or less compete to make the best game that incorporates this year's competition theme: the *Millennium Development Goals*[2]

[1]http://www.imaginecup.com/
[2]http://www.un.org/millenniumgoals/

Another difference will be the time frame available in which the product (the game) needs to be and will be developed. The deadline for Round 1 is March 1st 2009, so starting as late as the second half of the second semester is not an option. Instead, actual development will start on January 1st 2009, while the brainstorming and design phase has already begun. While the deadlines of the *Imagine Cup* must be upheld, it is by no means a requirement to continue to the next round, as that decision is up to an independent jury.

The remainder of this Orientation Report contains more information on when we are going to make what, using which tools, and how we are going to make it. Section C.2 describes what it is we are going to make, whereas section C.3 describes how we plan to make it. Then, our organizational structure and required required resources are described in section C.4. Finally, section C.5 contains some information on quality assurance.

C.2 Project contract

Project environment Microsoft organises a worldwide competition in multiple disciplines named the Imagine Cup. We are competing in the Game Development competition and going to build a game on the XNA[3] platform which runs on the PC and Xbox 360 console. The game is about the theme of the competition: "Imagine a world where technology helps solve the toughest problems facing us today." Those problems are the eight United Nations Millennium Goals.

Project goal Microsoft says: "The United Nations has identified some of the hardest challenges in the world today in its Millennium Goals. This year the Imagine Cup uses these ambitious challenges as a guiding light to inspire change all over the world. We can't wait to hear about the innovative solutions you create!" [4] So the main goal is to come up with inspiring solutions to realize the Millennium Goals. We decided to make our game in such a way that the player is becoming aware of the severity and complexity of the problems by playing and wants to help śolvingt́hem in a fun way. Since you can do *anything* in a game we can use simple solutions to the challenges in the game.

Project assignment Create a game that inspires and may offer new solutions to realize the Millennium Goals. The game has to be build on the XNA platform and has to run on a PC with XNA Game Studio 3.0.

Deliverables Round 1: A playable game demo about the theme of the competition: the Millennium Goals (or at least one of them). A short game summary (150 words maximum) of the game including its premise, unique gameplay features and how it addresses the 2009 Imagine Cup Theme. Gameplay instructions to explain how the player should play the game.

Round 2: A 100% playable game with one complete level and near final graphics based on the game demo of round 1. A game summary (maximum 200 words) like in round 1. 3 screenshots that accurately capture the gameplay and style of the game. Gameplay instructions like in round 1. A short game video (maximum 3 minutes) that depicts the functionality and gameplay features of the game.

Round 3: A presentation (maximum 30 minutes) to a panel of judges and participate in a Q&A session with the judging panel.

Requirements and limitations The game must be build with Microsoft Visual Studio 2008 + Microsoft XNA Game Studio 3.0 and must be developed for operation on Windows XP SP2, Windows Vista and/or Xbox 360.

The content of the Game must address a social cause connected to the 2009 Imagine Cup Theme: "Imagine a world where technology helps solve the toughest problems facing us today"

[3]XNA is a recursive acronym for *X*NA's *N*ot *A*cronymed
[4]http://www.imaginecup.com/Competition/mycompetitionportal.aspx?competitionId=21

The content of the Game must be equivalent to an Entertainment Software Rating Board (ESRB) rating of "E" for Everyone[5]. Meaning the content of the game may be suitable for ages 6 and older. Titles in this category may contain minimal cartoon, fantasy or mild violence and/or infrequent use of mild language.

The judges must be able to play the Game, as specified in the XNA Game Studio Toolkit.

All entry materials must be submitted and presented in the English language.

C.3 Approach and planning

Game Development is an environment that is constantly evolving. Whilst you are working on something, you get new ideas and think "Now would it not be cool if we also add this or that". Therefore we did not see it fit to put all our requirements in a Gantt chart and make a fixed planning that we would be changing every week.

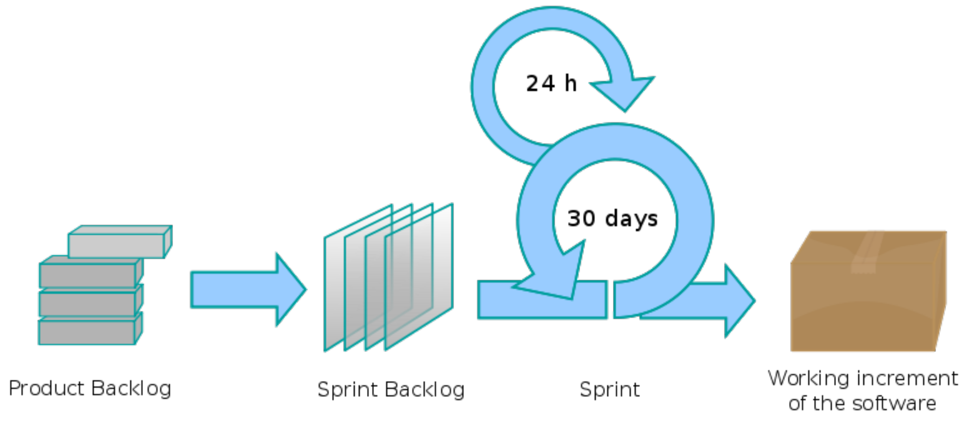


Figure C.1: The Scrum Process

That is why we decided to go for a more agile planning method called *Scrum* [6]. Scrum provides us with an iterative incremental framework that allows us to develop in a flexible way. If we have a new idea, we add it to our product backlog; a list of features we would like to add to the game that increases as we go. Like a rugby scrum, we are supposed to put our heads together at the start of each week and assign Business Values to each new task in the backlog and prioritize our backlog this way. The top listing tasks in the backlog will be our Weekly Sprint, meaning that those selected items will be the parts that we will be working on that week.

Because we evaluate our backlog each week we can rapidly see if something is in a state that is good enough for the moment, so we can go further to the next major task. By developing in this way, we do not get stuck on nitpicking on non-critical features and make the core of the game very fast.

We plan to have a development freeze the week before the first deadline, so we can completely test the game for bugs and get a perfectly functioning product.

[5]http://www.esrb.org/ratings/ratings_guide.jsp
[6]http://en.wikipedia.org/wiki/Scrum_(development)

C.4 Project organization

Organization Although this might not end up begin the best choice, we have decided not to make any one person responsible for any one part of the project, at least not initially. While we are developing, one person might implement, design or otherwise create a substantial part of one aspect of the product, but in the end we are all responsible for what has been made.

Resources Our workplace will be Building 35, where we hope to be able to reserve room DW0.011 for the duration of the project. This room has six PCs, all of which should already have the proper hardware, and a large whiteboard. The necessary software still needs to be installed on the PCs, however. For a list of software we think we need, see Appendix C.6.1.

While the software is not yet installed in Building 35, we are allowed to use one PC from the Computer Graphics department of the TU Delft. This PC does not have a fixed location inside the EEMCS building, so a suitable location where the three of us can work on the one PC has to be found on a day to day basis. Whenever possible, we will use room HB12.100, where a large whiteboard is located as well.

Reporting Because we will participate in a multi-round competition, the larger interim reports will be the deliverables for the various rounds. What reports will be handed in for the various rounds is described in section C.2.

We will also provide regular status updates to the coordinator of this BSc project, Dr.ir. A.R. Bidarra. Every other week we will have a meeting with our mentor, Remco Huijser, and every week in between a mailed update will be provided to him. Finally, we intend to maintain a blog[7], although it will not contain much detail about what we are doing initially.

C.5 Quality assurance

Games are always difficult to test due to their dynamic environment. There are a lot of different variables active in the virtual world that might influence the performance of an object. Therefore we plan to use *spiking*. This means that we create different small worlds and demo's to test each unique part before unleashing it into the real world. For example, if we plan to test the interaction of the player with an enemy we will create a level where you have only the player, the enemy and a floor. This minimizes the chances of error opposed to when you play in an interactive world where a lot is happening at the same time.

Besides the ordinary testing to make the game as bug free as possible, we also plan to have our game play tested by children of different elementary schools located in Delft. We want to do this for two different reasons. First, the children are our target audience and we can only do so much to imagine how they think and what they like. Secondly because we are constantly busy with the game we are gonna take things for granted or get used to small defects. Letting the children play with our game will provide us with fresh insight with what is wrong with the game and make us look at the game from a new perspective.

[7]http://www.teamwarp.nl/

C.6 Appendices

C.6.1 Necessary Software

At the workstations

- Microsoft XNA Game Studio 3.0
- Microsoft Visual C# 2008 Express OR Microsoft Visual Studio 2008
- TortoiseSVN OR SVN command-line client
- Drivers for Xbox 360 controllers

At home

- Microsoft XNA Game Studio 3.0
- Microsoft Visual C# 2008 Express OR Microsoft Visual Studio 2008
- TortoiseSVN OR SVN command-line client
- Drivers for Xbox 360 controllers
- Corel Paint Shop Pro OR Adobe Photoshop
- Autodesk Maya OR Autodesk 3DS Max OR Blender

Appendix D

Orientation Report

This document describes the orientation phase of our Bachelor Project. Unlike the usual Bachelor Projects, however, that phase never really ended, since the design of a game cannot be set in stone right from the start. Indeed, it is practically impossible to orientate oneself on everything that could possibly make it into the game beforehand. That is why this document has been continuously updated throughout the project.

Because the aspects that make a game are diverse, this report has been divided into several sections. Section D.1 describes the many ideas for we had for the story and setting of the game. Section D.1 follows with describing the artistic styles we wanted to accomplish. Finally, section D.3 describes what techniques we researched for use in our game.

D.1 Story and Setting

For the orientation of the story and setting we looked at different games we like, varying from first person shooters to sim-like games. One of the first games we came up with was a mix between FarCry 2 and the Metal Gear Solid franchise. Doctors Without Borders would travel trough Africa to help the people, but their supplies would be taken away by soldiers of an evil warlord. The player, a villager of where the supplies were taken, wants to help his sick family and plans a plot to regain the vaccines.

The player would need to sneak at night and stay out of sight of the guards in different situations. Although the concept was a bit ambitious, we did not dispose it because of its complexity, but for a whole other reason. After we read the rules of the Imagine Cup it was stated the game needed to have an E-for-Everyone rating, which meant there needed to be no violence in the game.

Another game we looked at was ActRaiser. In this game there were two parts of gameplay: at first you needed to defeat the evil beings of a part of your land in sidescrolling action kind of game. After that you needed to guide the people to the now cleaned area in a sim-city fashion.

This idea is based around the idea that in every area of the map live a few inhabitants in a bad situation. The goal of the game then would be to improve the situation in those areas and spread the population around the world to learn from each other. To improve each area the player would have to fulfill different tasks on improving water, food, education etcetera and this would be done by the means of fun mini-games. Like building pipes for an irrigation system or piecing tiles of different kinds of food together on a piece of land so it all will fit. The population rate then will depend on the resources of water and food.

But the reason we did not choose this concept was that previous winners of the Imagine Cup created Sim City like games, and we wanted to create something new and innovative. We decided to do this because we wanted to stand out of the crowd, and first and foremost create a game that would be fun and then mix that with the Imagine Cup goals in a subtle way. Instead of saying: 'This is what you need to do, or the world will have a problem!', we wanted the people to experience the problems there are in the world. By

Figure D.1: Screenshot taken from the game FarCry 2 by Crytek

doing so we wanted to motivate them to start their own initiatives. After creating some more concepts, we finally came up with the one we are currently making: Xylos.

D.2 Style

We plan to create a game with a cartoon shaded visual style with colorful vibrant colors. You can see in figure D.3 and figure D.4 some examples of what we want to achieve.

We wanted to create the characters of the game to be in the same semi-realistic style of the games shown above. But unfortunately, due to the lack of us finding an artist, we had to let that dream go. We decided, after deliberation with our mentor Remco Huijser, for a less realistic looking style, that would be more easy for us to create with our limited Maya skills. We plan to use basic geometric shapes, which will give a playful toy-like look to the game. Texture wise we want them to be just a single diffuse color which we think will give a cool and unique look when we mix it with a cartoonesque shader.

Even after we found an artist we still kept the geometric shapes. Mainly because this would be simpler for the artist as well, since we only teamed up with him just a month before the deadline. But we did give the artist some artistic freedom, and although it did not look like what we had in our heads, we were still happy with what it had become.

D.3 Techniques

Because all of us had already made a game at least once in the previous year during another project, most of the orientation happened then. It would be impractical, if not impossible, to include orientation documentation about that project in this report[1].

Because we still did not know everything, and never will, about the technology behind games, the orientation continued in this project. This section documents most of that orientation, and at least all orientation done on a larger scale. The sporadic searches for how something should be implemented have not been included, as it would only bloat the document unnecessarily.

D.3.1 XNA vs Cannibal

Very early on in the process, we came to a crossroads. Either make the game in 'pure' XNA, or use the Cannibal Engine. On the one hand, using XNA meant we could get information from a large community, and there are a lot of tools and examples to be found across the Internet. On the other hand, we have only used Cannibal in our previous project(s) and support would be a few hundred meters away.

After careful consideration we decided to keep using the Cannibal Engine, not in the least because of the awesome features the engine already had and the ones they were implementing at that moment. The other important reason was the practically on-site support they could give, and this way we could also help to improve their engine.

D.3.2 Procedural content generation

Procedural content generation was one of the first things we looked at for technical solutions, not in the least because we did not have an artist yet. Another reason was that two members of our team had to research these techniques for another course anyway [1]. Several types of content could be generated procedurally, but not all of them could be applied to our game.

Texture generation We did not think we would find serious use for this, and therefore did not investigate further. However, much later in the project, the team of Cannibal Game Studios incorporated texture generation into the infinite terrain they made.

Model Generation We did want to get in some procedural model generation. We found several awesome examples of procedurally generated trees using L-systems, and since the setting of our game required trees, we could not not at least try to implement this. In the end, the result was not as visually stunning as the examples, but it fits the style of the game nonetheless.

Terrain Generation Although theoretically it would be possible to procedurally generate the terrain of the world in our game, we chose not to. We liked to have some more control over what the terrain looks like then just some variables then can be tweaked. Also, it would have been made more difficult to use in combination with the Cannibal Composer we used.

Level Generation We initially did not think this would apply to our game. However, as we progressed, we though it might have been nice to use this to scatter some vegetation and other props around the world. Unfortunately we already had very little time left at that point for the other things we wanted and had to do, so we never got round into researching this option more thoroughly.

[1]The project referred to is the MKT4 Games project, held in the third and fourth quarter of the second year. One of the project goals was to create a game using the Cannibal Engine and XNA, in collaboration with several artists and game designers from the HKU.

Puzzle Generation It was highly unlikely we could use procedural generation to generate puzzles, because of the time travel mechanic we wanted to implement. Puzzles making use of that mechanic need to be thought out carefully, as to prevent the player from getting stuck and to prevent impossible puzzles.

Quest Generation Due to the nature of the game, and more importantly the time available to us, procedural quest generation would not make it into the game. Aside from that, quest generation is still too much in its infant stages to be implemented properly.

D.3.3 Sound

Because we had a game where time could be reversed, it would add to the effect if the music also played backwards if time was reversed. After digging through the various features of the XACT platform[2], we came up with nothing that could reverse playback of the music. We then thought it was also possible to use a second track, containing the reversed version of the first track, and play that when time was being reversed.

While this solution was the most feasible, we still came to a problem we wanted to solve: the music you hear being reversed should be the reversed version of the part of the track you just heard. Otherwise you would always hear the same when going back in time. Unfortunately, the music is not streamed, or at least not in such a way that the position of the track can be set, so it was impossible after all.

Although it would probably have been better if we used the second option, but did play the reversed track from the same place over and over again, it did not make it into the game. It was not because we did not want it in the game, but we simply did not have enough time, nor gave it a high enough priority for it to take precedence over other things. In the end, we only had a background loop playing during gameplay, and that was it.

[2]XACT, the Microsoft Cross-Platform Audio Creation Tool, is a tool that comes with XNA. This is the usual and easy way both 2D and 3D music is added to an XNA game.

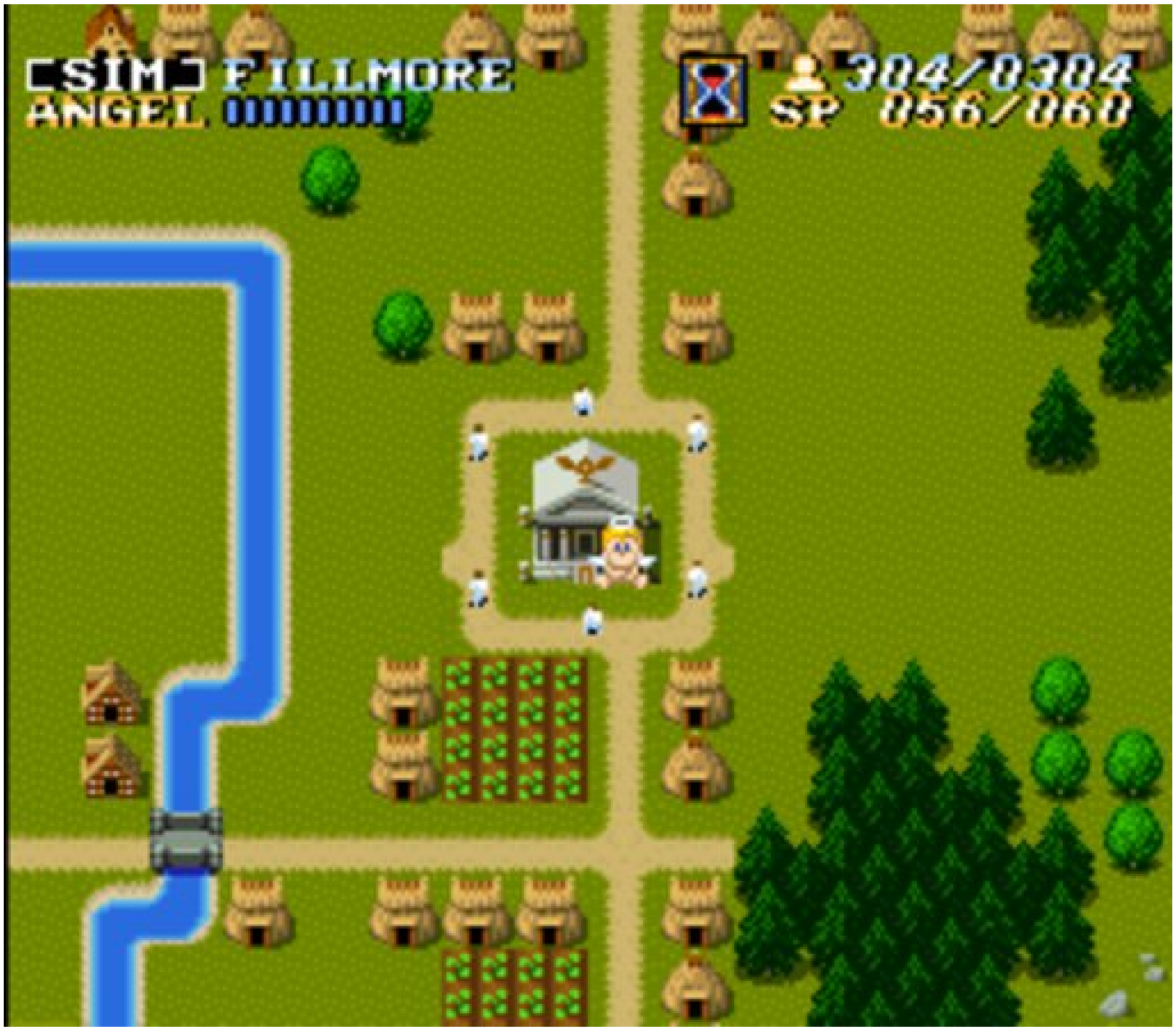


Figure D.2: Screenshot taken from the game ActRaiser by Quintet

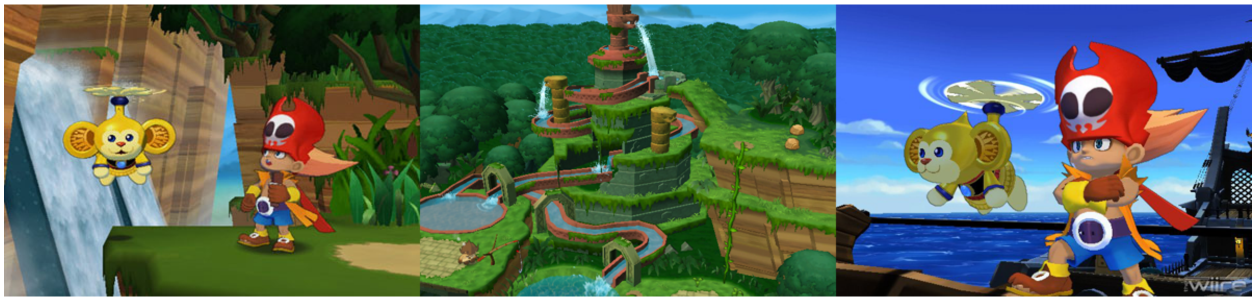


Figure D.3: Screenshots taken from the game Zack & Wiki: Quest for Barbaros' Treasure by Capcom

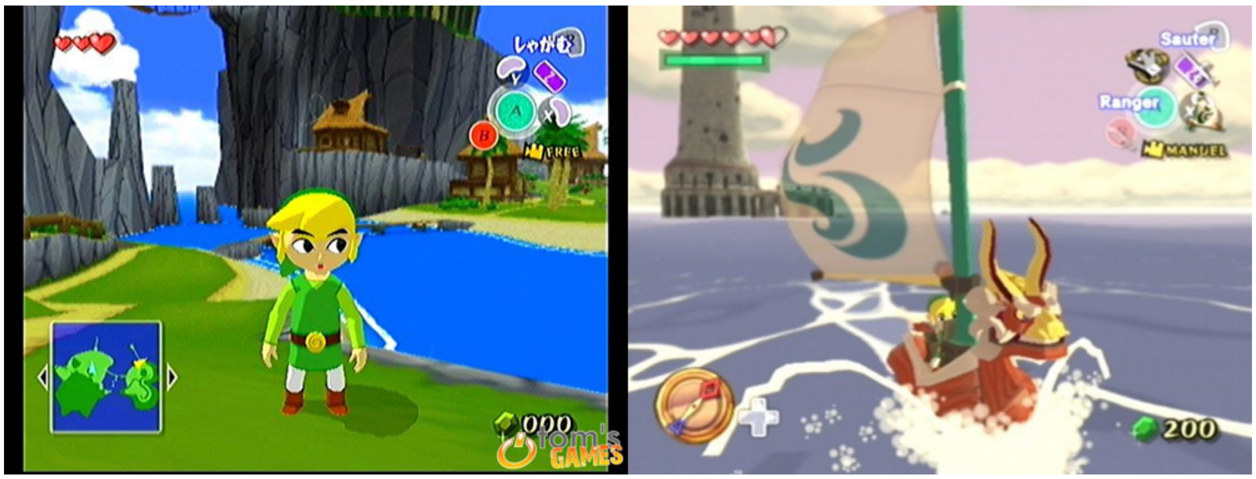


Figure D.4: Screenshots taken from the game The Legend Of Zelda: The Wind Waker by Nintendo

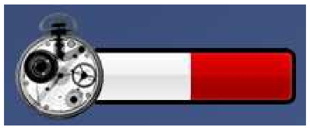
Appendix E

Imagine Cup Deliverables

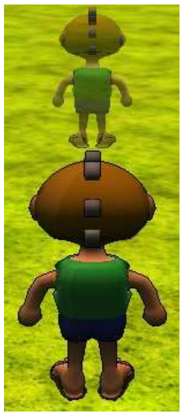
In the next few pages some of the deliverables for the Imagine Cup are included. First up are the Gameplay Instructions for Round 2. After that is the Game Document we made for Round 1.

Xylos – Gameplay Instructions

Time travel

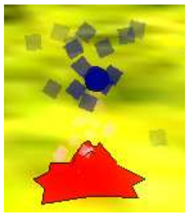


By holding down the left trigger or R key, you can travel back in time using the time gadget Kron. Time traveling costs energy, and the energy Kron still has left is displayed in the top-right corner. The energy automatically regenerates, but only when you're not traveling back in time.



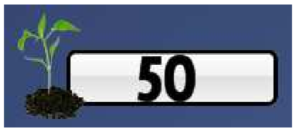
After you've traveled back in time, you will see a previous version of yourself performing your actions from a few moments ago. While there are previous versions of yourself present in the world, some penalties are applied to the energy:

- Traveling back in time again costs more energy
- Slower regeneration of the energy

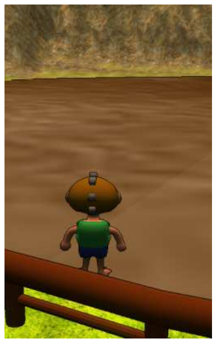


If you are low on energy, you can pick up an energy boost to instantly replenish some energy. You cannot pick up an energy boost when Kron has full energy.

Using other gadgets



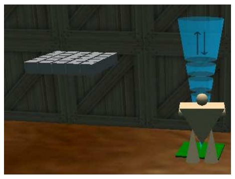
As explained above, the time gadget Kron can be used using the left trigger or R key. The other gadget to plant seeds and grow a tree instantly must be used differently. This way the player always has the possibility to travel back in time.
To use the equipped gadget, press the X button or the G key.



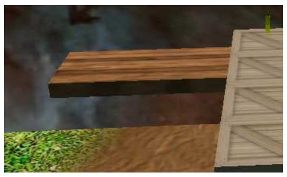
If you want to use the tree gadget, you will need a seed to plant, and stand on a spot where a seed can be planted (the brown dirt). If you fulfill these requirements and press the X button or G key, a tree will be grown instantly. How many seeds you have is shown in the top-right corner. Planting more trees gives the people of Warf more food they need.

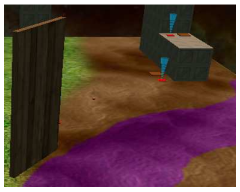
Buttons, platforms, bridges & force fields

A button can be pushed by simply walking on top of it. However, using objects that respond to the pressing of buttons aren't always that simple:



- Elevator platform. At first sight, this one is quite simple: you push a button and the platform goes up. However, this means you have to stand on both the platform and the button to actually use the platform.





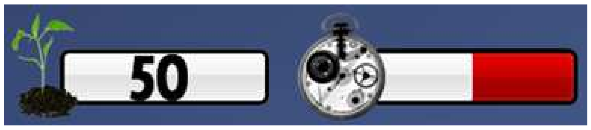
- Extending bridge. This one is an easy one; push the proper button(s), and the bridge extends.

- Drawbridge. This one's also easy, as all you have to do is push the proper button(s) to make it turn.

- Force field. This is a red transparent obstacle you cannot walk through, but that can be disabled by pressing the right button.

Buttons have a display above them indicating what kind of object is linked to it. If a button belongs to multiple types of objects (not in the demo level), the icon will change over time. Keep in mind that some bridges, platforms, or other objects that respond to buttons may require more than one button to be pressed.
Another thing to note is that, as one would expect, the bridges and platform also retrace their actions when you travel back in time.

Goal of the Demo level



The goal of the demo level is to give the population as much food as possible, using the resources found in the level. By talking to the villagers you can find out if they need more food or not.
Please note that there will be no message when the goal is met. You are free to experiment with the game afterwards.

Controls chart

Keyboard	**Xbox 360 Controller**	**Action**
W	Left thumbstick up	Move forward
S	Left thumbstick down	Move backward
A	Left thumbstick left	Move left
D	Left thumbstick right	Move right
Q	Left trigger	Turn left
E	Right Trigger	Turn right
Left Ctrl	Left shoulder button	Change into FPS-like controls. (only lasts while the button is held)
R	Y	Rewind (hold button to go back longer)
G	X	Use currently equipped gadget
H	B	Toggle gadgets
N	Press right thumbstick	Eat a seed
Space bar	A	Jump
Up arrow	Right thumbstick up	Move camera up
Down arrow	Right thumbstick down	Move camera down
Left arrow	Right thumbstick left	Move camera right
Right arrow	Right thumbstick right	Move camera left
n/a	D-pad up	Select health gadget (if available)
n/a	D-pad down	Select no gadget
n/a	D-pad left	Select tree gadget (if available)
n/a	D-pad right	Select water gadget (if available)
Escape	Back	Exit game
Enter	Start	Pause game
Middle mouse button	Right shoulder button	Look behind
Scroll wheel	n/a	Zoom camera

TEAM WARP
KICKING GAME DEVELOPMENT INTO HYPERDRIVE

Paul Brussee - Nick Kraaijenbrink - Bas van Sambeek

Game Document
Xylos

Version 1.0
Delft, February 2009

CANNIBAL GAME STUDIOS

TU Delft Delft University of Technology

Contents

1. Introduction 3

1.1 Short Game Description 3

2. Imagine Cup Theme 4

2.1 Goal 4

2.2 Target Audience 4

2.3 Effectiveness 4

2.4 User Experience 4

3. Technical aspect 5

3.1 Software Used 5

3.2 Target Platform 5

3.3 Innovation 5

3.4 Graphical Aspect 5

3.5 Extensibility 6

3.6 Planning 6

3.7 Business plan 6

3.8 Testing 7

3.9 Known Defects 7

Appendix A – Game Level Chart 8

1. Introduction

We are Team Warp, consisting of three computer science students of the Delft University of Technology. We're all very interested in developing video games for quite a few years.
When we became aware of the Imagine Cup we saw this as a way to do our share of bringing change to the world. And why shouldn't we do this in a fun an entertaining way while we're at it? That is why we've decided to join the game development track, to bring the Millennium Development Goals under the attention of the new generations!
More information about us can be found on our blog: imaginecup.basvansambeek.com.

1.1 Short Game Description

On a day Xylos, a young boy who is the main character of this story, finds a strange device in the sand while he is working. Xylos and his fellow Warfians are being suppressed by Worf, ruler of the planet Warf. Worf has taken away all the resources from the people so that the people are dependent on him. All the Warfians are working for Worf in order to get some food and shelter. The device Xylos finds turns out to be a time traveling AI gadget called Kron who tells our hero how Worf took the throne a long time ago. Xylos and Kron then decide to travel back in time to prevent that this event ever happened. Kron also allows Xylos to go back in time for almost a minute which enables Xylos to be at different places at the same time and accomplish tasks that cannot be done otherwise. For instance to defeat the guards of Worf.

Once they've regained Deme (the food gadget) from the past they head back to the present and use it to plant some seeds, which can be found at various places in the world, to grow some trees that produce food for the Warfians.
Because the children needed to work for Worf they didn't go to school and never learned to read. During the search for seeds you will meet the giant Macro Ploo, who was a teacher before Worf took the throne. He'd love to teach the children again how to read, but unfortunately he doesn't have any books anymore. During the pursuit of quest Xylos will find some books which he brings to Macro. The books also provide whereabouts information of the new gadgets.

Because Xylos has acquired Deme before Worf could get its hands on it thanks to Kron, Worf had to think of something else to gain control over the people. That's why he stole the water gadget Osei instead in the altered past and polluted the rivers. So Xylos and Kron need to go back in time again!
Going back and forth in time Xylos and Kron gather the total amount of three gadgets: food, water and health. The locations of all the gadgets are revealed by the children Macro Ploo teaches. Once all the gadgets are successfully retrieved you will face Worf in an epic boss fight to free all Warfians from Worf's oppression once and for all.

2. Imagine Cup Theme

The eight millennium goals still aren't fully achieved today and in order to help the UN reach those goals we want to raise awareness by the means of this game. This game incorporates the most goals in one way or the other. We decided not to include all the themes since this would make the game too complex and difficult to play.

2.1 Goal

The goal of this game is to raise awareness for the Millennium goals in a fun and educational way. We all know about some of the sad things that are taking place in third world countries, but although this is reality we decided this wasn't really suitable to put in a kids game. Therefore we decided to let the game take place in a fictional world. This gives us the ability to still bring the message across but in a cheerful way. We hope to inspire the children to keep taking care of the world and its inhabitants so that, when the Millennium Development Goals have been achieved in six years, they will keep pursuing them and take them to the next level.

2.2 Target Audience

We're aiming at a young audience whose characteristics are:

- People with affinity to fantasy, adventure and role-playing games.
- Experience with playing videogames (but not hardcore gamers)
- Love movies and cartoons
- Don't like violent games
- Love puzzling and creativity
- People who like a colorful environment in games.

2.3 Effectiveness

By playing this game the player is meant to:

- Become aware of the problems that still exist in the world today.
- Inspire the people around them to make a change in the world.
- Inspire the people that after achieving the MDG the development doesn't stop. For example we can aim to reduce child mortality by ninety percent instead of the two-thirds it is now.

2.4 User Experience

The entry level of this game will be low so that children with even no gaming background at all will be able to experience our amazing story and learn about the Millennium Development Goals. They will experience a great adventure together with our heroes in a story that keeps them on the edges of their seat and keep wanting them to play more.

3. Technical aspect

3.1 Software Used

The software we've used so far is:

- Microsoft Visual Studio 2008 Express
- Microsoft XNA 3.0
- Cannibal Engine
- Autodesk Maya 2008 (student license)
- Corel Paint Shop Pro X (student license)

Some models are downloaded from www.turbosquid.com.

3.2 Target Platform

Although this game currently only works on the PC, we're aiming to have the Xbox 360 as our core platform.

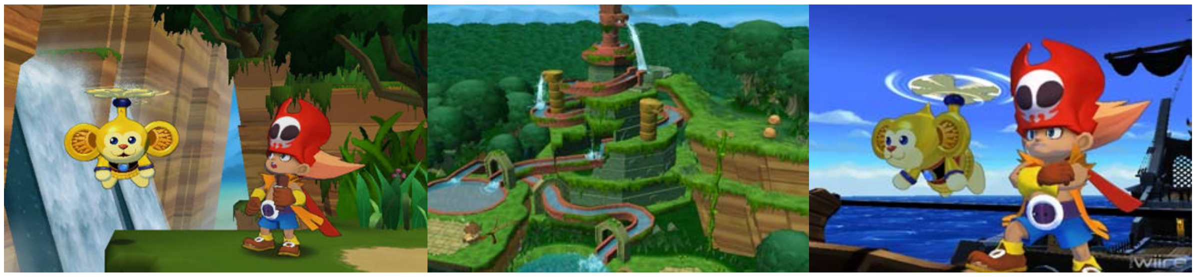
3.3 Innovation

This game is unique because it's one of the few games that incorporates time traveling in a 3D environment. It has been done a few times in 2D worlds like in the Xbox Live Arcade game Braid, but 3D is quite rare (at least we didn't find any games with this technology). By the means of time traveling, we are offering our audience (literary) a new dimension of puzzles.

3.4 Graphical Aspect

For this demo we focused on the technical and game play aspects of the game. For the next round we are aiming to create a game with a cartoon shaded visual style with colorful and vibrant colors. Given below are some examples of what we want to achieve.

Zack & Wiki



The Legend of Zelda: The Wind Waker

3.5 Extensibility

We would like to extend our game with the following features:

- Add mini games when using a gadget like the water or health gadget. This can be an opportunity to increase the learning aspect and variation of the game even more.
- Since we also target children of a lower age we want to localize the game, since we assume their understanding of the English language is quite low.
- If time permits we want to add multiple storylines affected by the choices you make in the game, like getting gadgets in different order and meeting different people.

Also, the concept leaves room enough to make (a) sequel(s). A possible (but maybe a bit cliché) storyline would be that Xylos leaves the gadgets (save for Kron) with his people, and he and Kron venture into the world to help and free other populations on Warf.

3.6 Planning

For the planning of time and resources we use Scrum. The Scrum framework enables us to develop our game in an iterative and incremental way by means of so called "sprints". New ideas and requirements are constantly prioritized according to their "business value" meaning what's more important at the moment gets done first. This way we can perform the most pressing tasks faster and reach higher productivity.

3.7 Business plan

We want to distribute the game on Xbox Live Marketplace for a small fee since the children need to be able to afford it with their pocket money. Also by making the price low, more people are able to buy it and it's better to have ten sales worth €5,- than one sale of €10,-. On top of that, because this game benefits to the Millennium Development Goals we want to donate 20% of the earnings to a cause that supports the MDG.
We are still investigating how the game can be published on the PC.

Because this game is focused on children we want to make it as fun as possible. We want to achieve this by not only making a game but also adding some promotion around it. We can do this by for instance creating an online community featuring our heroes where people will be able to discuss the game and the Millennium Development Goals. Another way is to let the game take part of an educational campaign on elementary schools, like a themed week with assignments designed around our heroes.

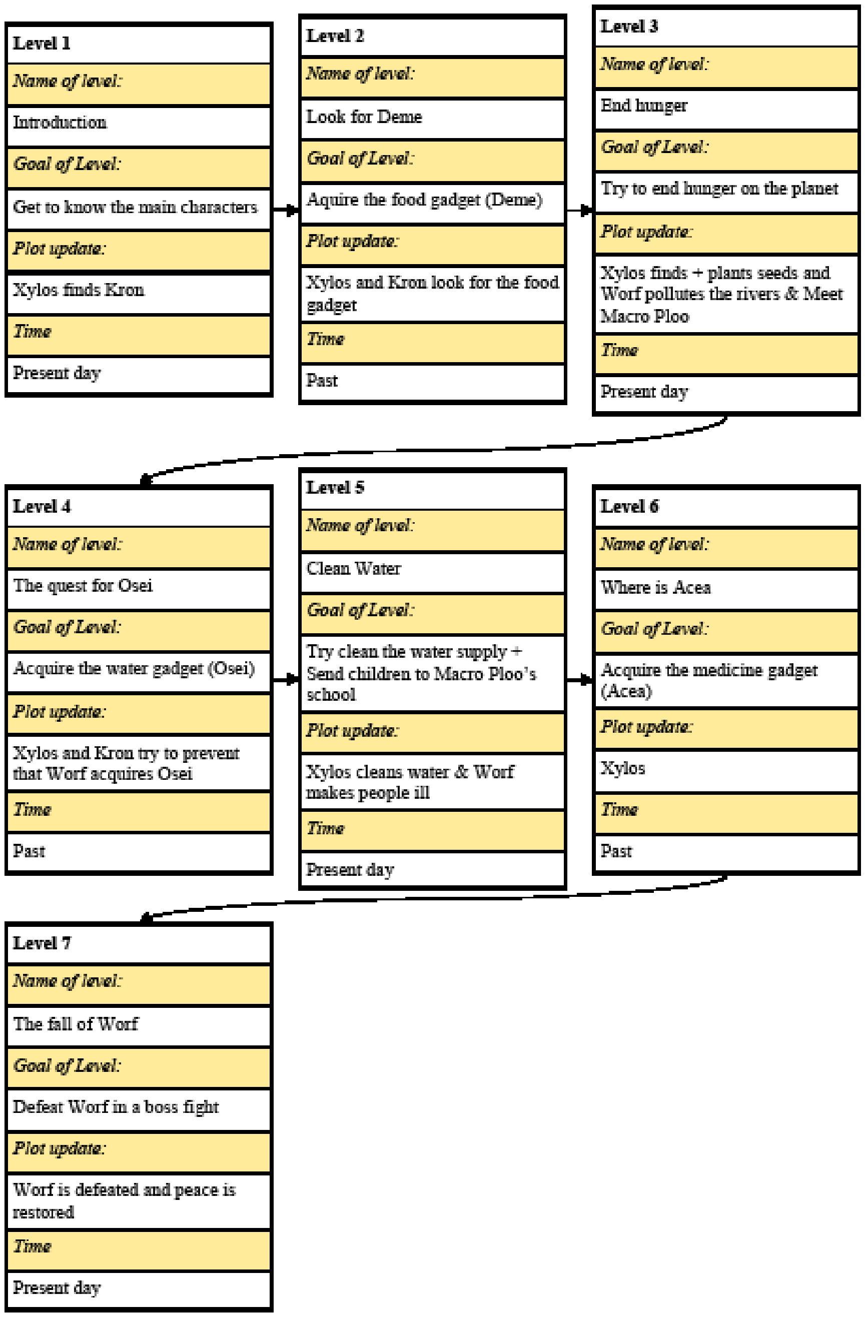
3.8 Testing

Besides the regular testing to make our game as bug free as possible, we plan to approach one or more elementary schools in Delft, and maybe a high school, for some play testing. We will let the children or teenagers play our game and incorporate the feedback gathered during these sessions in our game. We plan to do this at least twice on different schools before round 2 and hopefully more.

3.9 Known Defects

At the moment, one cannot do anything and everything in the world. Most of this is to prevent paradoxes from happening. This hasn't worked completely, since there are still some scenarios where the previous versions of the player do impossible things. For example, when one stands on an elevator platform for a while, rewinds, and then makes the platform go up while the 'time-clone' is on the platform, the clone stays where it is and the platform moves right though him. We hope to fix this for the second round, or we might restructure the game so that it's OK to move previous versions of oneself.

Appendix A – Game Level Chart



Level 1
Name of level:
Introduction
Goal of Level:
Get to know the main characters
Plot update:
Xylos finds Kron
Time
Present day

Level 2
Name of level:
Look for Deme
Goal of Level:
Aquire the food gadget (Deme)
Plot update:
Xylos and Kron look for the food gadget
Time
Past

Level 3
Name of level:
End hunger
Goal of Level:
Try to end hunger on the planet
Plot update:
Xylos finds + plants seeds and Worf pollutes the rivers & Meet Macro Ploo
Time
Present day

Level 4
Name of level:
The quest for Osei
Goal of Level:
Acquire the water gadget (Osei)
Plot update:
Xylos and Kron try to prevent that Worf acquires Osei
Time
Past

Level 5
Name of level:
Clean Water
Goal of Level:
Try clean the water supply + Send children to Macro Ploo's school
Plot update:
Xylos cleans water & Worf makes people ill
Time
Present day

Level 6
Name of level:
Where is Acea
Goal of Level:
Acquire the medicine gadget (Acea)
Plot update:
Xylos
Time
Past

Level 7
Name of level:
The fall of Worf
Goal of Level:
Defeat Worf in a boss fight
Plot update:
Worf is defeated and peace is restored
Time
Present day

Bibliography

[1] P.W.G. Brussee, N. Kraayenbrink, F. Post, and S. Sijbrand. Survey of procedural game content generation. In *Seminarium IN3305. Student Papers, dinsdag 24 maart (ochtend)*, pages 45–57, 2009.

[2] Rob Bryanton. *Imagining the Tenth Dimension*. Talking Dog Studios, Inc., 2006.