

BACHELORTHESIS:

WIZARD WARS – ANDROID GAME DEVELOPMENT

AUTHORS:

Jon Sætheren Lande

Anders Einar Hilden

Ole Marius Kohmann

Date: 27.05.2011

ABSTRACT

Title:	Date : 27.05.11	
English:	Wizard Wars – Android game development	
Norwegian:	Wizard Wars – Android spillutvikling	
Participants:	Ole Marius Kohmann	
	Anders Einar Hilden	
	Jon Sætheren Lande	
Supervisor(s)	Øyvind Kolloen	
Employer:	Jayson David Mackie at the Game Technology Lab, Gjøvik University College	
Keywords	Game, Android, Mobile, OpenGL, Two-player	
Number of pages: 181	Number of appendix: 8	Availability (open/confidential): Open
<p>Short description of the bachelor thesis:</p> <p>This project implemented a two player game on mobile devices running the Android operating system. The game, named Wizard Wars, used network communication between the two players, allow them to play on separate devices at different locations. The primary test device for the project was HTC Desire mobile phones.</p> <p>The game uses the touchscreen interface on the phone for all game input, including a gesture recognition system for the main interaction during gameplay. The game involves manoeuvring around an arena and casting spells at the opposing wizard.</p> <p>Starting this project, the group had minimal experience with game programming. After many hours of studying game technology and the Android system, trying, failing and debugging code, we have learned a lot about working in a team and creating larger systems. We have spent 3 years learning several topics and in this project we have used these topics in a practical situation, which has given us a new view and insight into these topics.</p>		

PREFACE

We would like to give a special thank you to our supervisor, Assistant Professor Øyvind Kolloen, for his help with the documentation during our project.

Our employer, Jayson David Mackie, along with Simon McCallum at the Game Technology Lab, Gjøvik University College has been very helpful with our code and game design issues.

We would like to thank Gjøvik University College for their good cooperation during the project.

Ole Marius Kohmann

Anders Einar Hilden

Jon Sætheren lande

TABLE OF CONTENTS

1	Introduction	11
1.1	Project description	11
1.2	Goals.....	11
1.3	Scope.....	12
1.4	Target audience	12
1.6	Our background and expertise	12
1.7	roles.....	14
1.8	Work methods	14
1.8.1	System development model.....	14
1.8.2	How did we work?	15
1.9	The report	15
1.9.1	Organization	15
1.9.2	Terminology.....	15
1.9.3	Practical	15
1.9.4	Layout	16
2	Requirements specification.....	17
2.1	Game description	17
2.2	Functional requirements.....	17
2.2.1	Roles	17
2.3	Use case diagram	18
2.3.1	General use case descriptions	19
2.4	Use cases in a game	21
2.5	Detailed use case descriptions.....	22
2.6	Operational requirements	24

2.6.1 Performance	24
2.6.2 Ease of use	24
2.6.3 Security	24
2.6.4 Network connection	24
2.7 Limits	25
2.7.1 Language and environment.....	25
2.7.2 Database	25
2.7.3 Operating system.....	25
3 Design	26
3.1 Starting the app.....	26
3.2 Starting the game.....	28
3.3 The Game loop	29
3.4 Updating the world	31
3.5 Scenegraph.....	32
3.6 Rendering the world	33
3.7 the hud	37
3.7.1 Movement	38
3.7.2 Actions	38
3.7.3 Health and mana bars.....	38
3.8 Animating the player.....	39
3.9 Casting a spell.....	40
3.10 Collision handling	42
3.11 Loading world and game objects	44
3.12 Player Rotation.....	46
3.13 Mana Regeneration	47

3.14 Network.....	48
3.15 Game server	48
3.16 Optimizations	49
3.16.1 Memory allocation	49
3.16.2 Threading.....	49
4 Design Decisions	50
4.1 Action System.....	50
4.2 Update manager vs. event handler/process manager	51
4.3 The Gameserver	54
4.4 Client Networking	55
4.5 Input.....	55
4.5.1 Movement	55
4.5.2 Gestures/Action.....	56
4.6 Rendering	56
4.7 Game loop.....	58
4.8 Collision	59
5 Implementation	61
5.1 Naming convention	61
5.2 Organization of the code	61
5.3 Design patterns	63
5.3.1 Singleton	63
5.4 no.hig.rag.UpdateManager.....	64
5.4.1 UpdateManager.....	64
5.5 no.hig.rag.controllers.....	65
5.5.1 Controller	65

5.5.2 TranslateController	66
5.5.3 RotateController	68
5.5.4 AnimationController	69
5.5.5 ActionController	70
5.6 no.hig.rag.actions.....	71
5.6.1 Action.....	72
5.6.2 SpellDefault.....	73
5.7 no.hig.rag.gui	75
5.7.1 Gestures.....	75
5.7.2 Movement	75
5.8 no.hig.rag.network.....	76
5.8.1 NetworkSender.....	76
5.8.2 NetworkReciver	76
5.8.3 Combined.....	77
5.9 no.hig.rag.collision	78
5.9.1 CollisionManager	78
5.9.2 Cell	78
5.10 no.hig.rag.human.....	79
5.10.1 Human	79
5.10.2 Player	80
5.10.3 NetworkPlayer	80
5.11 no.hig.rag.scenegraph.....	81
5.11.1 Scenegraph	81
5.11.2 Nodes.....	81
5.11.3 Node	82

5.11.4 TranslateNode	82
5.11.5 RotateNode.....	83
5.11.6 GeometryNode	84
5.11.7 TextureNode	84
5.11.8 GroupNode	85
5.11.9 SwitchNode.....	86
5.11.10 Initialization of nodes	87
5.11.11 Serialization	88
5.12 no.hig.rag.datastructures	89
5.12.1 Vec3	89
5.12.2 Vec2	89
5.12.3 RotateVector.....	89
5.12.4 VecCell	89
5.12.5 BoundingBox.....	89
5.13 Tools.....	90
5.13.1 Parsing game objects and creating scene graph file	90
5.13.2 Main.....	90
5.13.3 BuildInitialSceneGraph	90
5.13.4 ColladaParse	90
5.13.5 GameObject.....	90
5.14 Server	91
5.14.1 Communication	91
5.14.2 The components.....	92
5.14.3 The database tables.....	93
5.15 3rdparty.....	94

5.15.1 OpenGL	94
5.15.2 Collada	94
5.15.3 Android	95
6 Testing.....	96
6.1 Test strategy.....	96
6.2 Testing tools/methods	96
6.3 Game server	97
7 Closing.....	98
7.1 Discussion of the results	98
7.1.1 The result	98
7.1.2 Discussion	100
7.2 Work methods	102
7.2.1 Scrum	102
7.3 Afterthought	105
7.3.1 Do it yourself or use middleware?	105
7.3.2 Merging systems.....	105
7.3.3 The OpenGL/Android headache	106
7.3.4 Action system	106
7.3.5 Collada vs .OBJ	107
7.4 Conclusion.....	109
8 Appendices	113
Appendix 1: Project diary.....	115
Meeting 20.05.2011 - Feedback Øyvind.....	115
Appendix 2: Worklogs.....	135
Work log for Ole Marius Kohmann.....	146

Worklog Anders Einar Hilden	152
Appendix 3: Status reports	155
Appendix 4: Scrum meetings	157
Sprint 7 (29.03.2011 - 04.04.2011)	157
Appendix 5: Daily scrums	161
Appendix 6: First Technical description What we need	168
Appendix 7: Project Plan (Pre project report)	170
5 Table of contents□	170
6 1. GOALS AND BOUNDARIES	171
1.1 Background	171
1.2. Goals.....	171
1.3. Boundaries	171
7 2. SCOPE.....	173
2.1. Project description	173
2.2. Scope.....	173
8 3. Project organization.....	174
3.1 Employer and supervisor	174
3.2. Responsibilities and roles.....	174
3.3. Group rules and routines	174
3.3.1 Group rules	174
3.3.2 Routines	174
3.4 Resources	175
9 4. Planning, meetings and reporting.....	176
4.1. System development model	176
4.2. Meetings	177

4.3 Status reports.....	177
10 5. Quality assurance	177
5.1 Testing.....	177
5.2 Code meeting.....	178
Appendix 8	180

1 INTRODUCTION

1.1 PROJECT DESCRIPTION

Wizard Wars is an action game made for mobile phones running Android. The game will allow two players to play competitive against each other on separate phones. The idea is to have wizards fight each other, in real time, where the setting is a battle in an arena between two players.

1.2 GOALS

Our main goal was to make an entertaining game, which was technically functional, so that it could be played. This meant that both the client and the server had to be operational at the end of the project.

Another important goal for the project is the learning aspect. We are excited about learning game programming, and programming for Android, as well as improving our overall programming skills.

Writing a game is a complicated task, requiring knowledge from many of the courses we have had already, mentioning a few: Programming, algorithms, operating systems, computer architecture, WWW-technology, program development, system engineering, math and physics, with more.

We found it interesting to do a project that involves a lot of the courses we have had, since none of our previous project have done this in particular. We are using a lot of time playing games ourselves, so to learn the technology behind games seemed very exciting.

1.3 SCOPE

We realized that first making a game engine, and then implement the game play logic, would take up a lot of time. Since the main goal was to make the game technically functional at the end of the project period, we had to keep the artwork to a minimum. The main focus was to implement the features needed to make the game playable.

1.4 TARGET AUDIENCE

The target audience for the game is basically anybody interested in gaming, with an Android based phone, although we imagine the game appeals mostly to people at the age of 10 - 30.

The target audience for this report is the projects' external sensor, our supervisor and employer, other students, and others who might be interested in game programming. This means that we don't expect the reader to have any prior knowledge specifically about game programming. However, a general understanding of programming and computers is expected.

1.6 OUR BACKGROUND AND EXPERTISE

Out of the three members of the group, Ole Marius was the only one with experience from game development. He had previous work experience as a game designer, and some knowledge to game technology from a few courses here at school. The two other group members had no experience with game programming at all.

The Android system was new to all of us. However, Android apps are programmed in Java, which Anders Einar and Jon had some experience with, from the program development course at school. Ole Marius was new to Java, so he needed some time to adjust to it.

The server was programmed by Anders Einar with PHP and MySQL, which he knew well.

We used a lot of time to learn game technology, and understanding the Android system.

As time went by, we saw that because of our limited experience in game programming, we needed more time on each part of the project than we expected. Especially the development of the game engine became time consuming. We realized that we had to cut back on some features.

We decided to drop the feature where players can choose their opponent. Instead, the server finds two players looking for a new game, connecting them, and the game starts on both phones. The two phones need to be connected to the same local network to play against each other, and to the internet to get connection with the server. The help section and view statistics did not get implemented either. The results of the project are discussed under 'Discussion of the results', in the 'end' section.

1.7 ROLES

- Ole Marius Kohmann has been group leader, lead designer, and artist.
- Jon Sætheren Lande has been project manager, and responsible for game object loading.
- Anders Einar Hilden has been Scrum master, and responsible for the server and network.
- The projects employer was Associate Professor Jayson Mackie at the Game Technology Lab, Gjøvik University College.
- Our supervisor was Assistant Professor Øyvind Kolloen.

1.8 WORK METHODS

1.8.1 SYSTEM DEVELOPMENT MODEL

Since we had minimal experience with game programming, we would probably need to change our design, ideas and time limits through the development process. This meant that we wanted to use an agile method as our development model.

When looking at the different methods we narrowed it down to 3 candidate methods; RUP, Scrum and Extreme Programming (XP).

Two of the group members used RUP as method in the systems engineering course and concluded afterwards that it is best used in large teams and big projects. Even though XP could be used, we felt that Scrum was more suited to our needs, so Scrum was our preferred choice of development model.

We think that Scrum as the development model was an excellent choice for our project. Its rules and guidelines are easy to understand, and they ensure good documentation and communication within the group. Scrum is suitable even for a small group like this. More about this under 'Work methods' in the 'Closing' section.

1.8.2 HOW DID WE WORK?

Most of the project work has been done at our group room, A030. We shared this room with another group. This has worked great.

In addition to the scrum meetings, we have had three meetings with our supervisor, discussing the progress and our report. Jayson Mackie and Simon McCallum at the Game Tech Lab at the school have been very helpful with game design- and coding questions. We have had a lot of smaller meetings with them. The most important meetings during the project period were logged, and are found in Appendix 1.

1.9 THE REPORT

1.9.1 ORGANIZATION

This report is designed based on the guidelines given from the Gjøvik University College for Bachelor project reports.

We have chosen to let the design, and the discussion around our design decisions, be separated. The design part can at time become quite big and we want to make sure that both topics are easily found. We have also added a title in our document called “Afterthought”. Here we discuss topics about what we would have done differently, and problems that changed the course of our project during implementation.

1.9.2 TERMINOLOGY

As mentioned earlier, we expect that the reader of the report has a general computer understanding. When it comes to words and expressions we use that have to do with game technology, we will explain these the first time they are mentioned, before moving on.

1.9.3 PRACTICAL

The report is written in Microsoft Word 2010. For some of the documentation, we have used Google Docs in the process as well. Some of the figures are drawn by hand and scanned in, while others are made with Google Docs Drawing. For referencing, we use the Vancouver system.

1.9.4 LAYOUT

The report is written with the font type Calibri, size 12. The chapters and sub chapters are numerated, in three levels (e.g. ch.1.9.4).

2 REQUIREMENTS SPECIFICATION

The following requirements specification shows the requirements for the game before the development period started. This does not necessarily reflect what were implemented. Any deviations from the requirements are discussed in the discussion section of the report.

2.1 GAME DESCRIPTION

Wizard Wars is going to be a multi-player action game in 3D, for mobile phones running Android. The game will allow players to play competitive against each other. The idea is to have wizards fight each other, in real time, where the setting is a battle in an arena between two players. Players need to register and log in, in order to get an opponent and start a game.

The game will have a user interface where players can get suggestions for available opponents, and accept or reject these. Obviously the users can be challenged to a game themselves as well. When a player is challenged and the challenge is accepted, a new game is started.

The player will be controlled with the phones' trackball or optical sensor for movement. The players will cast spells at each other, in order to win the game. This is done by doing certain pre-defined gestures on the screen. When a spell is casted, the player's mana will decrease. If the spell hits the opponent, his health will decrease.

A game is won when a player is out of health. The players then get a message with the result of the game. The users' number of matches, wins and losses get stored, and users can see his statistics at any time.

2.2 FUNCTIONAL REQUIREMENTS

2.2.1 ROLES

The game will only have one type of user. The server handles all of its tasks automatically, we don't need an administrator.

2.3 USE CASE DIAGRAM



FIGURE 1: USECASE DIAGRAM FOR THE GAME.

Since there is only one type of user of the game, obviously every user can use all of the use cases. Some of the use cases from one user can start a use case at another user, like 'choose opponent' will trigger the use case 'answer opponent request' at the player that was chosen, and vice versa.

2.3.1 GENERAL USE CASE DESCRIPTIONS

USE CASES BEFORE A GAME STARTS

Following are short descriptions of each of the use cases in the game. Since the game only has one type of user role, all of the use cases are used by the same type of user, and thus the user is not specified.

Use case:	Manage player account
Goal:	Create a player account, and log in, in order to play.
Description	The user can register a new player profile, with a username, email and a password. A player account is needed to play against other people. Once registered, the user can log in and see the main menu. From here, he can start a new game, view statistics, or log out.

Use case:	View statistics
Goal:	The user can see his players' statistics.
Description	User can view statistics for his player. This should show number of matches played, wins and losses.

Use case:	See help
Goal:	Show user a help screen.
Description	Shows an explanation of how the game works. This should show what the different gestures look like, and explain how they work.

Use case:	Start new game
Goal:	Start looking for an opponent to start a new game with.
Description	When the user chooses new game, the game starts looking for opponents. If any available opponents are found, the user gets a list of these.

Use case:	Get opponent list
Goal:	Show the user all the possible opponents to choose among.
Description	User gets a list of available opponents for a new game. These are all

	other players who are also currently looking for an opponent.
--	---

Use case:	Choose opponent
Goal:	Choose an opponent to challenge for a new game.
Description	From the list of possible opponents, the user can challenge one of them to a new game. If this challenge is accepted, the match will start. If it's rejected, the user can choose a new opponent.

Use case:	Answer opponent request
Goal:	Either accept or reject a request for a new game.
Description	If the user has clicked 'new game', but has not yet challenged an opponent, he can get a request for a new game himself. This request can either be accepted or rejected. If accepted, the match will start.

Use case:	Offline game mode
Goal:	Start a new game in offline mode.
Description	This is a kind of single player mode, where the opponent is inactive. Here the user can learn how the game works and practice the gestures, even with no internet connection. Offline mode can be started without the user having to register or log in.

2.4 USE CASES IN A GAME

Use case:	Move player
Goal:	User can move his player around in the world.
Description	The user should be able to move his player around in the world. The movement is controlled with the optical sensor on the phone, or a trackball. Only moves along the ground will be possible, jumping will not be possible. If user tries to move into another object (wall, another player, etc), the move will not be allowed.

Use case:	Cast spell
Goal:	Cast a spell to hurt the opponent.
Description	To win the game, the players need to hurt the opponent enough. The way of doing this is casting different spells. Spells are casted by doing different gestures on the screen. The game will have minimum three different spells. The spells are going to be shown as small objects flying through the world, where it can either hit the opponent or miss. There has to be a clear sight to the opponent to get a hit. If the spell hits a wall it will get destroyed.

Use case:	See players info
Goal:	Show both of the players' health an mana status.
Description	While playing a match, both of the players can see their own and the opponents health status and mana status, each represented by a dynamical bar. These bars are shown all the time while in a game.

Use case:	Leave game
Goal:	Lets a user leave an active game.
Description	If, for some reason, a user wants to leave a game, and he presses the back button or home button, the active game should end for both players. This will not result in a win/loose for any of the players.

2.5 DETAILED USE CASE DESCRIPTIONS

We picked out two of the most important use cases, which are explained in detail below. Both of them happen inside an active game.

Use case:	Move player	
Goal:	User can move his player around in the world.	
Description:	<p>The user should be able to move his player around in the world. The movement is controlled with the optical sensor on the phone, or a trackball. Only moves along the ground will be possible, jumping will not be possible. If user tries to move into another object (wall, another player, etc), the move will not be allowed.</p> <p>Every time a player tries to move, a collision detection system is activated, to check whether the player collides with anything or not. If no collision is found, the move is allowed, and the player moves. If a collision is found, the player will either stop, or, if moving partially along an object (e.g. a wall), the player will slide along the object.</p>	
Type:	Essential	
Pre-conditions:	<div>1. New game started</div> <div>2. Player doesn't collide with another object</div>	
Post conditions:	Player has moved on both players' screens	
Detailed sequence of events:		
User:		System:
1. User tries to move his player		2. System detects a move, activates collision system.
		3. Collision system checks for collisions with other objects. No collision is found.
		4. Player gets moved to the new position.
Alternative sequence of events:		
User:		System:
1. User tries to move his player		2. System detects a move, activates collision system.
		3. Collision system checks for collisions with other objects. A collision is found.
		4. System calculates that the collision was direct into the other object.
		5. The move is denied, player doesn't move.

Use case:	Cast spell
Goal:	Cast a spell to hurt the opponent.
Description:	To win the game, the players need to hurt the opponent enough. The way of doing this is casting different spells. Spells are casted by doing different

	gestures on the screen. The game will have minimum three different spells.	
	The spells are going to be shown as small objects flying through the world, where it can either hit the opponent or miss. There has to be a clear sight to the opponent to get a hit. If the spell hits a wall it will get destroyed. The player is always facing the opponent, so the spells casted will always move towards him.	
	The accuracy of the gestures is found, and the better this is, the more damage the spell will do (if it hits the opponent).	
Type:	Essential	
Pre-conditions:	New game is started	
Post conditions:	a) Opponent is hit and damage is made. or b) Spell misses opponent, and gets destroyed.	
Detailed sequence of events:		
User:	System:	Opponent:
1. User moves player to the desired location		
2. Does the gesture which casts a thunder spell		
	3. The players' mana is decreased with a specific amount	
	4. System finds the accuracy of the gesture	
	5. Spell is "thrown" through air, and hits opponent	
		6. Opponent gets hit by the spell, and damage is made (based on the accuracy of the gesture)
Alternative sequence of events:		
User:	System	
1. User moves player to the desired location		
2. Does the gesture which casts a thunder spell		
	3. System finds the accuracy of the gesture	
	4. Spell is "thrown" through air, misses the opponent, hits a wall and gets destroyed. Opponent is never affected.	

2.6 OPERATIONAL REQUIREMENTS

2.6.1 PERFORMANCE

We are going to make the code as efficient as possible. This means that our coding style sometimes have to differ from the generally recommended practices of programming, in order to increase the performance of the game.

Since our main goal is to get the game working on a HTC Desire, we should have more than enough hardware power to run a game like this. However, we still want the game to run as fast as possible.

We don't expect the server to get overloaded with users, but it should handle at least 50 users at a time.

2.6.2 EASE OF USE

We will try to make the game as user friendly as possible, without it affecting our goal of making the game functional at the end of the development period. To increase the ease of use, we want to make a help screen for guiding the user through how to play the game.

2.6.3 SECURITY

Security will not be a priority in our project. Considering the time limit of the project, we will rather focus on getting the game functional. On the registration of player, the password will be encrypted. We will not focus on preventing manipulation of game results. For further development of the game, this would have been an issue.

2.6.4 NETWORK CONNECTION

In order to register a player account, log in, and find opponents to challenge, the phone must be connected to the Internet, to get a connection to the server. The network traffic between the two players in a game will be handled by the server as well. However, if the phone has no Internet connection, there is still possible to play the game in offline mode, with an inactive opponent.

2.7 LIMITS

2.7.1 LANGUAGE AND ENVIRONMENT

All the parts of our project will be developed through the software development environment Eclipse. The project that creates the scene graph file will be a plain java project. The GUI shown before the game starts, and the actual game are both written in Java as well, for Android. The server will be written in PHP. For repository we have used Subversion (although we considered using GIT).

2.7.2 DATABASE

For our server system, we will use MySQL as our database engine. The database is used to store information about player accounts and games/matches.

2.7.3 OPERATING SYSTEM

We are going to develop the game for the Android platform, specifically Android 2.2. Although Android 2.3 is the latest version at the moment, we believe we will reach out to more people by supporting 2.2. Because of the time limit, we won't be able to support other versions or platforms.

We are limiting ourselves to developing the game primarily for the android phone HTC Desire. This is one of the most popular android phones on the Norwegian market. (1) (2)

3 DESIGN

3.1 STARTING THE APP

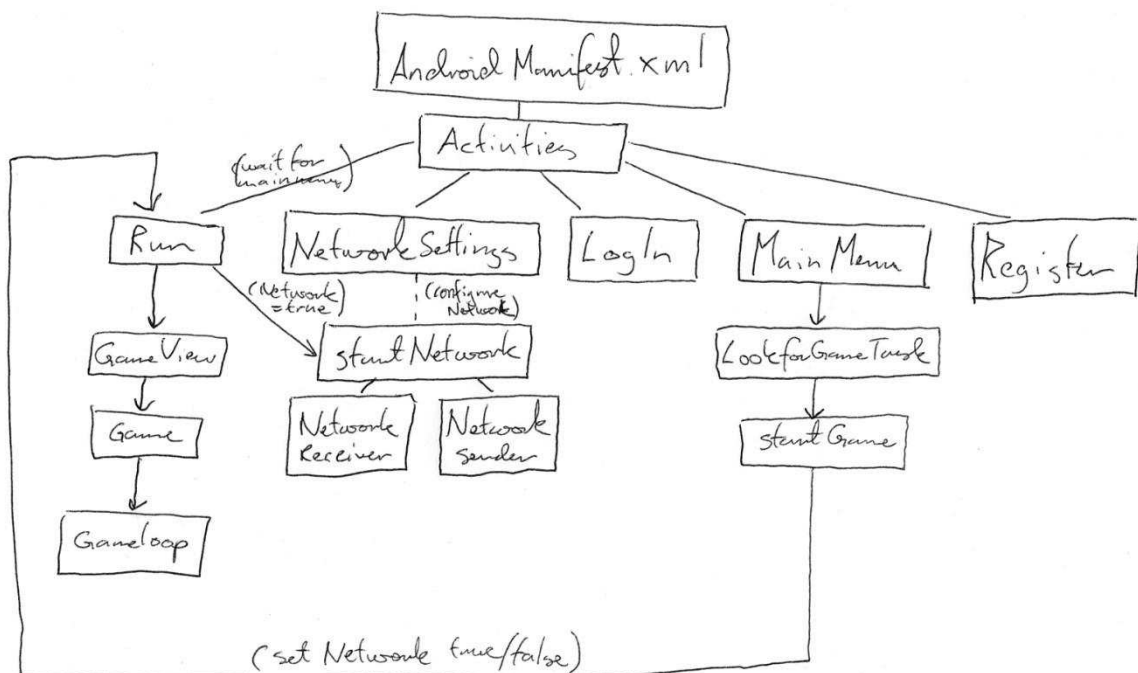


Figure 2: Showing how the activities are connected.

An app's activities are defined in the file `AndroidManifest.xml`. This file tells Android which activities that the app uses. What activity that starts first, is decided by the `<intent-filter>` tag inside an activity. The tag is used in our game to define `MainMenu` to start first. `MainMenu` then decides how `Run` is to be started.

`Run` can be started in two ways: With or without Network. In `MainMenu` you can start the game without network. This was used during the development period for debugging purposes, but it also serves as an offline mode of the game. When the player chooses "Log In" from the menu, he logs into the game frontend. There are 3 choices here, "New Game",

“View Stats” and “Logout”, what they do is self-explanatory. We did not finish “View Stats” so this feature is not in the game.

When pressing “New Game” the client sends a message to the server, telling it that it is looking for a game. The server will then try to match up the client with the next free client, who’s looking for a game. When an opponent is found, Run is started with network enabled and the other clients IP so both clients will connect to each other.

3.2 STARTING THE GAME

Starting the game

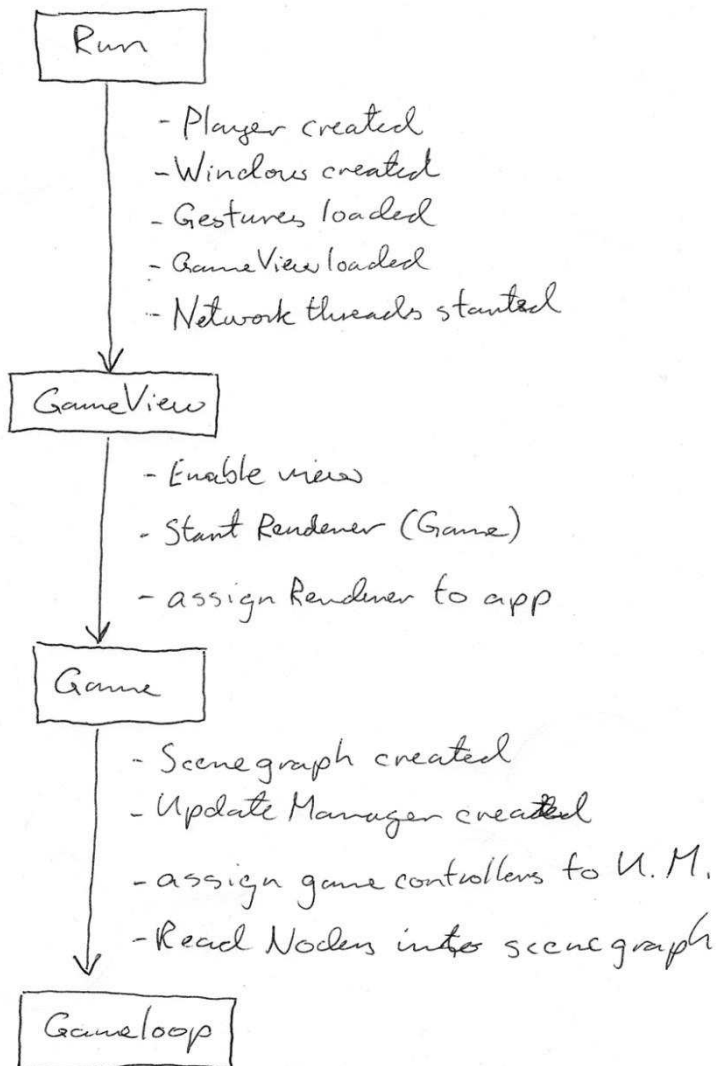


Figure 3: Shows what is started when Run is started and up to the gameloop.

When Run is started, the game initialization begins. First the Android system is told to create an OpenGL surface that we can render on, and to put a touch sensitive surface on top of this (so we can accept movement and gesture input). A gesture library is loaded so the Android automatically can recognize our prerecorded gestures.

Then we start the network threads (they communicate with Game), and a separate thread that checks the game state (whether anybody has won or lost).

We populate the OpenGL surface by one of our classes (Game). In this class we have combined both the rendering loop and update loop, and we call it the game loop.

3.3 THE GAME LOOP

The game loop is responsible for running/looping the game systems (input, updating, rendering, network) during the lifetime of the game.

A game loop on Android is a bit different from a normal computer platform game loop, much because there is already systems setup to do specific jobs. This means you need to order your game loop around this. An example of this is that touches on the screen (the primary input tool) are handled by an activity, without any easy way to pass these down to the render.

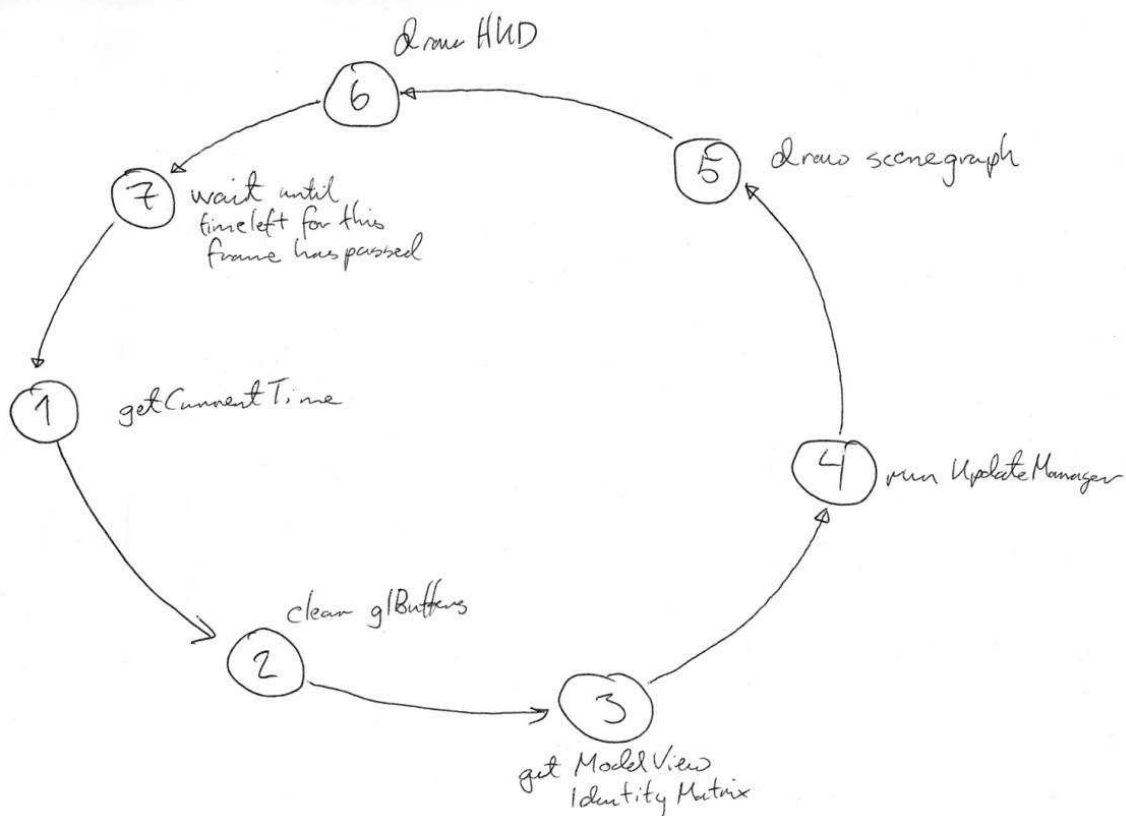


Figure 4: The gameloops main jobs.

In addition to the threads created by Android to facilitate the rendering, we have threaded our network sender- and receiver code. We would also have liked to have the renderer and update routine to be on separate threads, but did not get it into the project in time, so both are found inside Game.

Before the game loop is started, camera position, regeneration and similar are initialized, and then the world (including the players) is loaded from a pre made file.

Inside the game loop we first update the game (Figure 4). This includes movement, rotation, action, camera position and several other operations. Updates are explained better later in this chapter.

Once all updates have been completed we traverse the scene graph, to render the world. The scene graph is a graph that contains all the elements in our game world. More on this below, under 'Rendering the world'

On top of this rendered scene we draw the HUD (Heads Up Display), which includes navigation icon, health bars and mana bars. More on this under the chapter 'HUD'.

At the end of the game loop, we take a snapshot the time and compare this to one taken at the start of the game loop. This gives us the difference in milliseconds from when this frame started and when it ended. We then use this difference to set the loop to sleep for 1/30th second minus the difference. This ensures us that the game will not render more than 30 frames per second.

Parallel to the game loop we have a thread for sending, and another thread for receiving network packages. During the update cycle, new packages are created to be sent, and received packages are read, parsed, and used to update the status of the opponent.

3.4 UPDATING THE WORLD

We update the world by using an update manager and different controllers. The update manager is a class that makes sure that all of the controllers do their job, by iterating through them. A controller is a class with a specific job, for example to handle user input. Most of the controllers are connected to a specific node in the scene graph used in the rendering of the world, and their job is to modify this. More about the scene graph later.

Any Controller inside the UpdateManager knows what to do itself. It is preprogrammed to do a specific task. What the UpdateManager does, is providing each Controller with the current time and previous time (previous time being the timestamp for the last frame UpdateManager was run) so the Controller knows the timeframe which it needs to work within. For instance, the RotateController needs to know how much time has passed since last time it made a rotation, so it knows by how much it needs to rotate now. This gives a smooth rotation when looking at the rotating object over time.

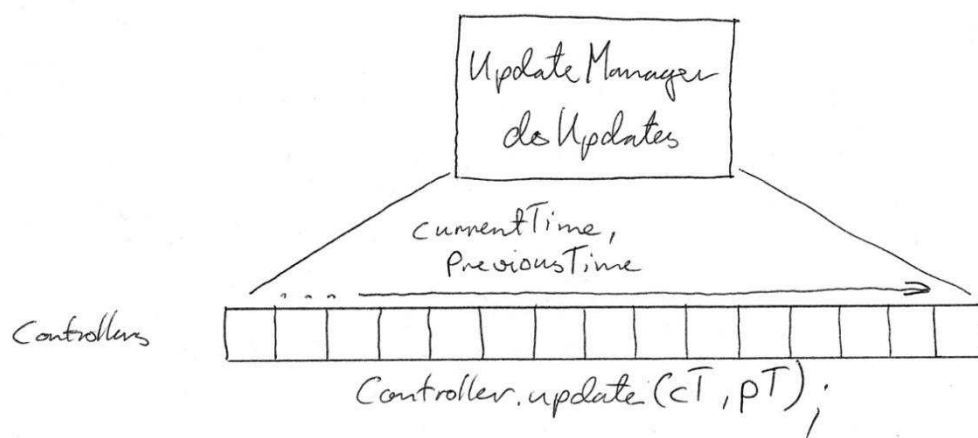


Figure 5: UpdateManager goes through all controllers, using time to call each update().

The Update routine is pretty straightforward. It is one function calling another function from all objects inside an array.

3.5 SCENEGRAPH

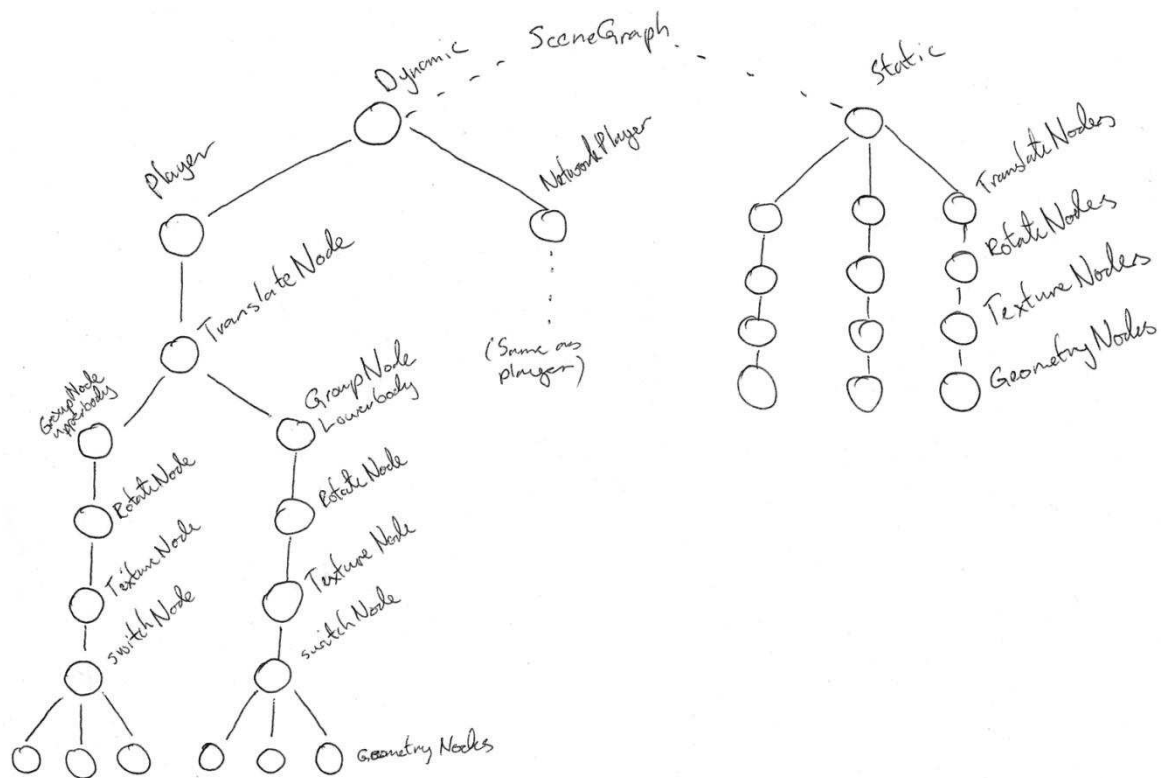


Figure 6: How a scenegraph is built up.

A scene graph is an acyclic graph which is made up of nodes. The scene graph itself is not a very complex data structure; the nodes contain all the code that defines the game world. This structure allows for easy and fast control over how a world should be rendered. The combination with the Controller classes, allow for complex behavior with minimal effort. This way of working with small changes to produce great results are the greatest advantage of using scene graphs for a game.

3.6 RENDERING THE WORLD

There are two things that are important for rendering the world. The first is making sure OpenGL gets properly configured. The second thing is to traverse the scene graph.

OpenGL is a state based system; this means that when you set a state to a certain mode, it will stay in that mode until you specify a different mode. Some states need to be set only once, but others need to be set to on, off or change its value once or several times per frame.

The value needed to be set once, are set inside `Run.onSurfaceCreated()`, rest are set in either `Run.onDrawFrame()` or inside nodes in the scene graph. The `TextureNode` draw function only enables a state in OpenGL that tells which texture to use. This texture will be used on all drawing from OpenGL until a new texture is specified, so a texture node needs to be used for every object drawn in the game.

For each frame the game loop is running, the scene graph is traversed. Every node performs its task for every frame. Since this is being done after `UpdateManager` has been run, all nodes have their data updated for the current frame.

There are seven different types of nodes and each one behaves differently (see below for description of each node type). The nodes are split into two separate groups: Static and dynamic. The static nodes are not moved after loaded, any node inside dynamic *can* move during frames. Differentiating nodes like this speeds up the update routine greatly, because if all nodes were to be dynamic, all nodes would have to have its own controller for translation and rotation. This is not a big deal for a scene graph the size we use, but we wouldn't need to add much before the update routine would take too long. As of now, the only dynamic nodes are the ones used in `Player` and `NetworkPlayer`.

Each node has a OpenGL command performed in its `drawTraverse()` function. The order of the nodes matter in terms of what result you are after. Below shows the different results of having a `TranslateNode` and a `RotateNode` swapping position in the scene graph:

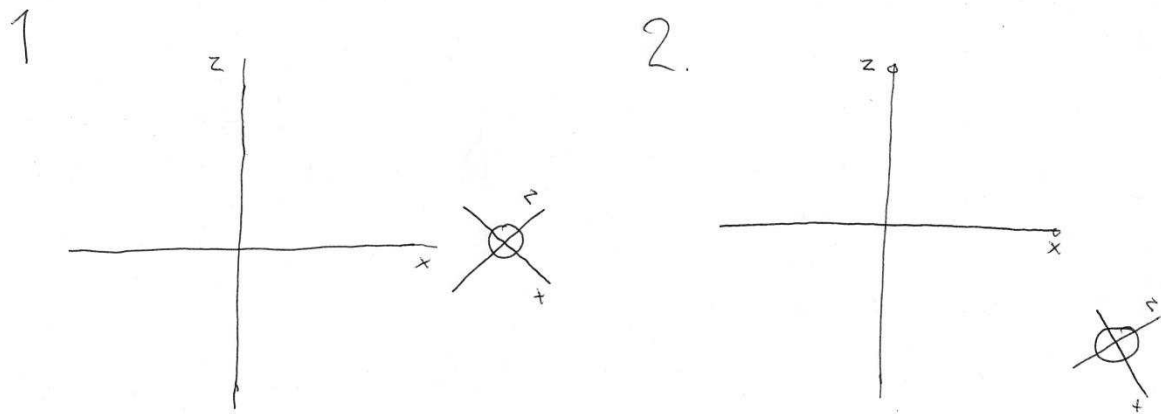


Figure 7: Different position of nodes in the scenegraph, cause different results.

In the first example, the RotateNode is before the TranslateNode. This means that the object is rotated by 45 degrees around center, while being in the center and then moved 5 units on the x-axis.

The second example shows the object being moved 5 units on the x axis first and then rotated 45 degrees around center.

For displaying models in the world, you first need to set the position and rotation, using a TranslateNode and a RotateNode. Then you need to add a TextureNode and at last a GeometryNode. You could also add several mode GeometryNodes to the TextureNode to have more than one model in the game, using the same texture.

The performDraw() function of the GeometryNode do the following: First it sets states in OpenGL that allows drawing objects the way we chose to draw them. It then transfers the data to OpenGL and asks it to draw it, when finished drawing, it disables all states that was enabled before.

We use the function glEnableClientState() to activate these states and glDisableClientState() to disable them. The states GL_VERTEX_ARRAY, GL_NORMAL_ARRAY and GL_TEXTURE_COORD_ARRAY tell OpenGL that we are storing data in arrays and that we use arrays for vertices, normal and texture coordinates.

Face Indices	Vertices	Normal	Texture Coordinates
1 2 3	x_1 y_1 z_1	x_1 y_1 z_1	s_1 t_1
4 5 3	x_2 y_2 z_2	x_2 y_2 z_2	s_2 t_2
4 2 1	.	.	.
5 6 2	.	.	.
5 4 3	.	.	.
2 4 5	x_n y_n z_n	x_n y_n z_n	s_n t_n

Figure 8: All arrays need to be index aligned for OpenGL to work properly.

It is imperative that the data being sent in these arrays are synchronized. This means, that any index in these 3 arrays, contain data about the same vertex! Some 3dmodel file formats do not support this, because 3d modeling packages don't care, but in OpenGL this is an absolute rule. By forcing this rule, OpenGL only need one index list for all vertices, all normals and all texture coordinates.

The objects are built up by triangle. A triangle is defined by 3 vertices, connected together.

After enabling the states, we set the triangle draw order to counter clockwise, letting OpenGL know which direction the triangles face. Any triangle that OpenGL draw, that is drawn clockwise, it then knows is facing away from the camera and is then not being drawn. This is something OpenGL does automatically.

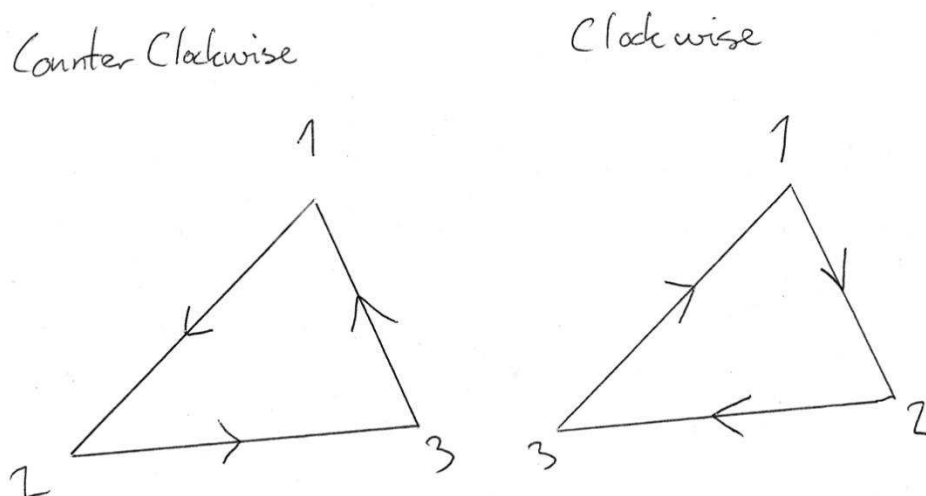


Figure 9: Counter clockwise or clockwise triangle drawing.

With the settings we have for triangle orientation. The triangle on the left would be facing the screen, and be drawn and the triangle on the right would be facing the same direction as the camera is and would not be drawn. This technique is called back-face culling. (3)

The data about the objects (vertices, normals, texture coordinates) are stored in directly addressed buffers, which the Java garbage collector cannot delete or remove.

What we do next, is assigning the buffers to OpenGL and identifies what kind of buffers they are.

Using the functions `glVertexPointer()`, `glNormalPointer()` and `glTexCoordPointer()`, we let OpenGL know what buffers to use and how to use them. Each function takes in as parameter how many variables there are for each instance in the buffer, what kind of variable it is and a pointer to the buffer.

When this has been set properly, OpenGL is ready to draw the vertices and `glDrawElements()` is called. This function takes as parameter what kind of 3d structure you want to draw, the size of the index list, what kind of variable the indices are, and a pointer to the index buffer.

Last thing to do is to disable the states we set first, and the triangles have been successfully drawn to the screen.

3.7 THE HUD

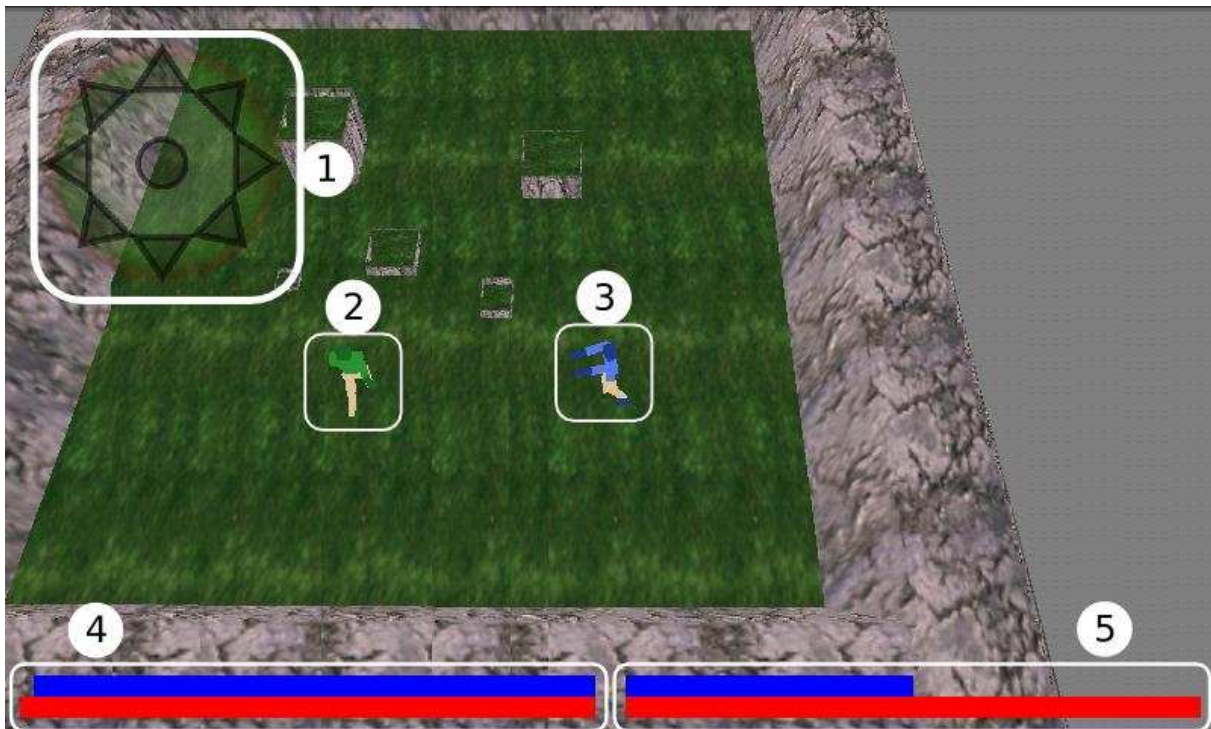


Figure 10: The HUD with health and mana bars for the player[3,4] and his opponent[2,5]. In the top left corner is the navigation icon[1]

HUD stands for Heads Up Display. It is basically an overlay on top of the game world displaying vital information and navigation controls. The GUI's primary mission is to translate the user's physical interaction with the device to game events (movement, actions). We have designed our game to accept two forms of input: Movement in the form of touch-events to the screen, (in a pre-defined area) and gestures drawn on the screen to start actions (casting spells).

3.7.1 MOVEMENT

Initially we wanted to use the trackball for movement. However, the trackball did not perform as expected. We therefore had to implement a special area on the screen that is used to detect movement. Touches inside the navigation icon [1] in the top-left corner of the screen are used instead of the trackball or other hardware solutions. These touches are then translated to display-size-independent values and passed on to user's player.



Figure 11: D-Pad

3.7.2 ACTIONS

Gestures are captured using a built-in library in Android that compares the user's finger movements over the screen area to prerecorded gestures. An event is triggered when the library finds a comparatively close match to the gesture. When a gesture is recognized and it is within the boundaries we set for how close a match needs to be, the action is sent into the game loop. There it is decided whether to discard it (another action might be in progress), to cache it, or accept it and start a new action.

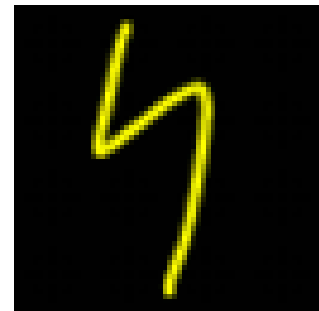


Figure 12: A Gesture

3.7.3 HEALTH AND MANA BARS



Figure 13: the Bars.

The GUI also provides the users with information needed in a game. Four dynamic bars that represent the users and opponents stats – health and mana - are drawn in the bottom of the screen. The player's bars are to the left and his opponent's to the right.

Using some math, we let the bars adjust and position themselves relative to the resolution and orientation of any screen.

3.8 ANIMATING THE PLAYER

Each player has two animation controllers; one for the upper body and one for the lower body. These provide a walking look for the player, along with an idle version, and one for when the player is casting a spell. For walking animations (walking or idle), the players' translate controller will ask the animation controller to play an animation, and when a player is casting a spell, the animation controller is asked to play the casting animation. An animation is an iteration of different frames/versions of the player in different poses.

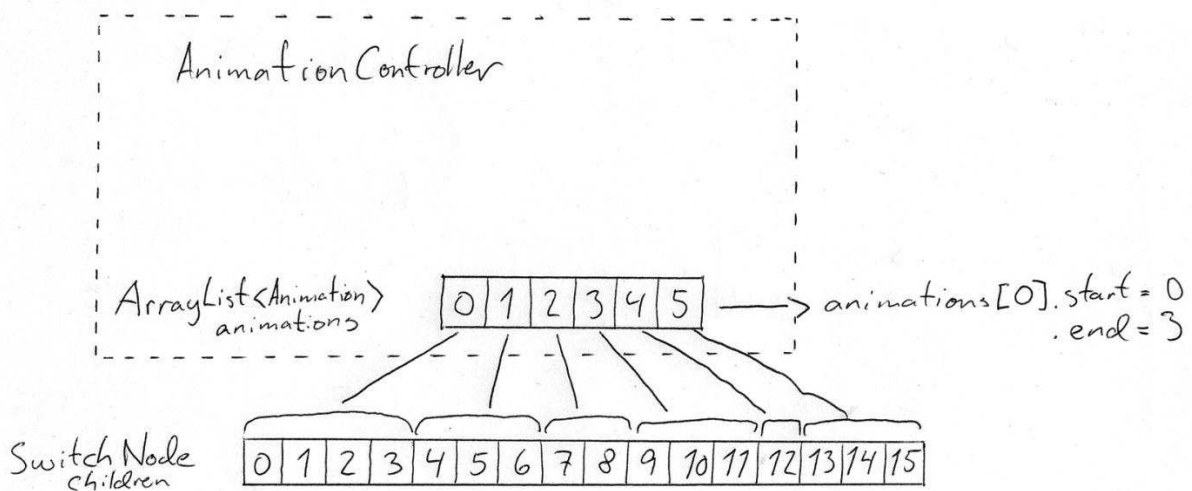


Figure 14: Animation Controller and its relationship to the SwitchNode.

The controller will then set this animation to be the current one. When `update()` is done it will then ask the current animation to iterate its frames. When rendering the screen, the switch node will then use the index set by the animation currently playing.

3.9 CASTING A SPELL

To hurt the network player, the player has to cast spells. A spell will decrease the network player's health bar, and decrease the player's mana bar.

Casting a spell is detected through gestures in android. It sends the id to the Player object if the Gesture is identified as a spell. Player then asks the ActionController to change current action to the one specified by the Gesture.

The ActionController then needs to check if it's allowed to activate that action and then activate it, or not. The only two rules as of now are to check area range, and whether an action is already being performed. Other rules to include would be: is the target visible? Did the player cancel the current running action? And so forth...

Once the rules are passed, it needs to call the action's update() and check for when it is complete. When an action is complete, it needs to kill that action and set itself to idle. This logic is quite fragile for bugs, and you can easily get into troubles where it breaks itself, or the Player does something that breaks it. If looking at the use of actions, from start to finish, the complexity lies in all the layers of code it is spread over; They are called from Gestures, sent to Player, started in ActionController, run in Actions and killed in ActionController. The hard part of solving this, is understanding where you put what parts of the logic.

We chose to solve this by limiting Gestures to only tell the player what action was called, which then tells the action controller what it *wants* to do. Then the action controller decides if that is legal or not. Once the action has been activated, it's the action itself that controls what happens. When the action is finished, it marks itself so the action controller knows it's finished and can remove it.

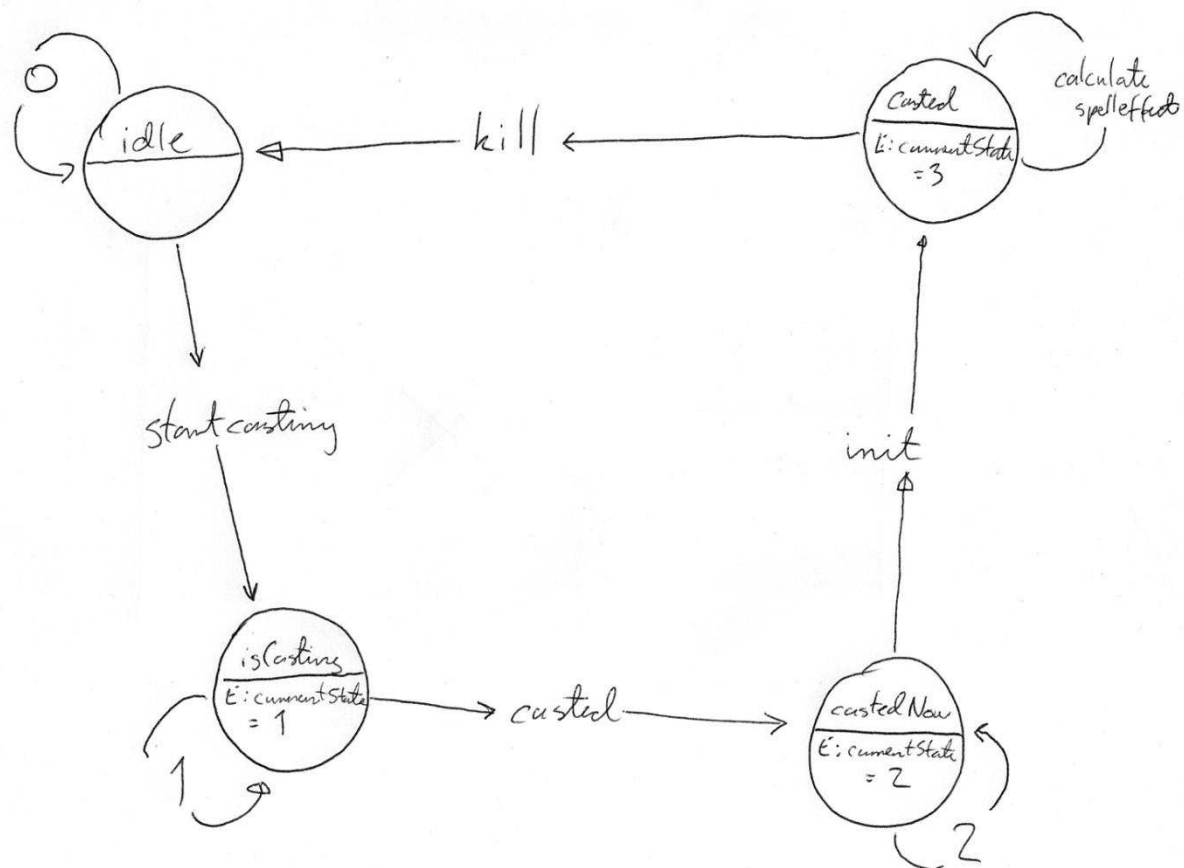


Figure 15: The internal FSM controlling the behavior of a spell.

When ActionController activates the spell, its set to 1, meaning it is currently being casted. When finished casting, the transition *casted* is performed, in our case this is just setting *currentState* to 2. In state 2 the spell is initialized and state is set to 3. During state 3 the effect of the spell is calculated on the target. Leaving state 3 calls the *kill* function and *currentState* is set to 0.

The function *calculateSpellEffect()* has 5 different methods of applying its effect. We chose to implement and test 1 of them, while the remaining 4 are implemented, but not tested. The effect we implemented is direct damage/healing done to the target. It affects the health of the target additive, using basic algebraic rules on adding two numbers together. This means that when, using signed variables, the target is to take 10 in damage; the variable is -10. If the target is to receive 10 health, the variable is 10.

3.10 COLLISION HANDLING

Almost every game needs some kind of collision handling. Without it, all of the objects would just float through each other. We need to check if there is a collision, and, if so, handle this collision in some way. We made a relatively simple system for detecting and handling collisions between game objects, such as players, walls and other objects the world might have.

There are many ways of representing an object in a collision system, called bounding shapes. What they all have in common is that they need to encapsulate the object completely.

We decided to use two-dimensional squares/boxes. These wrap the objects into a square on the ground. The red lines in figure 16 show the bounding boxes for the objects. The bounding boxes can never be rotated, so their edges are always parallel to the x- and z-axes.

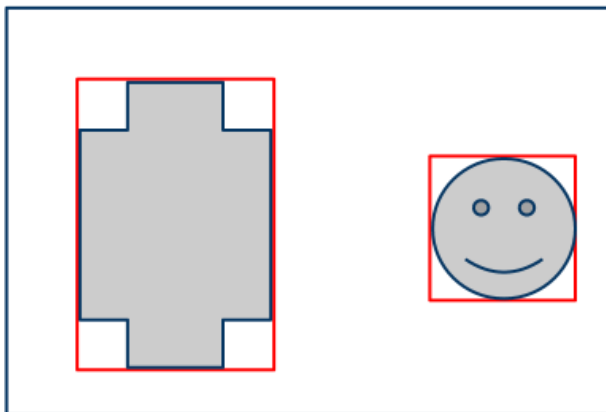


Figure 16: Bounding boxes represented by red lines

Every game object has its own bounding box. The object itself is never inside the collision system, it's only represented here by its bounding box. When the scene graph is read from file, the objects' bounding boxes are added to the collision system.

To avoid having to check for collisions between every single object in the world all the time, the world is divided into a grid of cells. The objects know which cell(s) they currently are in. This way, we only have to check an object for collisions against other objects in the same cell(s). Figure 17 shows how the grid looks like.

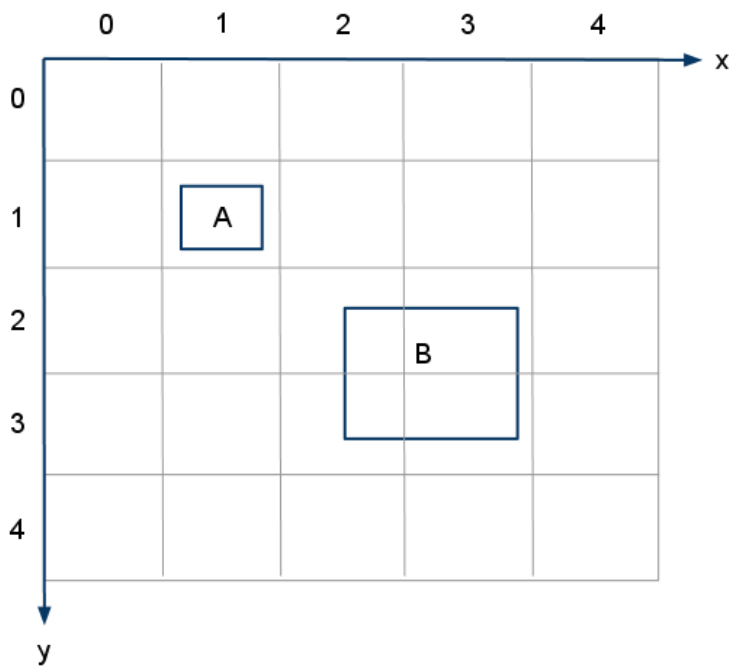


Figure 17: The grid of cells, holding the objects. Object A is in cell (1,1), while object B is in the cells (2,2), (2,3), (3,2) and (3,3).

It's the translate controller that moves both the objects and their bounding boxes. We check an object for collisions with other objects every time it tries to move. First, we move the objects bounding box to the desired location. If there was no collision, we move the object itself as well.

If a collision is found, we need to handle it. If an object tries to move directly into another object, the bounding box is moved back, and the object itself is never moved. If the move is more than 45 degrees away from the object we collide with, the object will move the desired distance in the direction parallel to the edge of the bounding box it is colliding with, but it will not move in the direction towards the object. This results in a sliding effect, where for example a player can slide against a wall. A collision is shown in figure 18. The arrow is representing the direction of the object colliding into another. The red line is the angle, α , between this direction and the edge of the other object. If this angle is below 45 degrees, or above 135 degrees, the colliding object will slide along the other object.

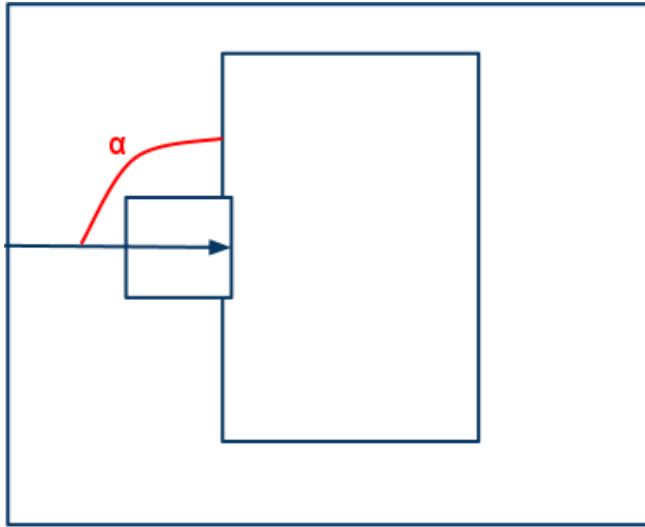


Figure 18: This shows a collision. α is the angle between the direction of the moving object, and the edge of the other object.

3.11 LOADING WORLD AND GAME OBJECTS

Our game object models are created with the 3D modeling program 3D Studio Max, and exported as XML files, in the Collada format (.dae files). These files contain the objects' texture-, normal-, and vertex- coordinates, and faces of vertices. We needed a way to parse these files, and transform them into OpenGL objects. We also needed an initial scene graph for when the game loads, that contains the world and all of its objects in their initial positions.

We could have done this every time the game starts, but we quickly agreed that this would slow down the performance. Therefore, we made a separate project called Create Scene Graph. This is where we build the initial scene graph. The game objects are parsed from the XML file(s), and added, one by one, to the scene graph. The scene graph is then written to a scene graph file (.sg-file). This file is included in the game. When the game starts, the initial scene graph gets read in from this file. As soon as you start playing, the nodes of the dynamic objects will obviously change.

For our bachelor project, we only had time to make one world/map, and therefore we got one scene graph file. In case of further development of the game, the plan is to make one

scene graph file for each map. As soon as this file is created, this project is no longer needed in order to play the game.

We could probably have found a Collada file parser online, but we decided to write one ourselves. This was mainly because the format is very comprehensive, and since we only needed a few of the elements from the files, we found it best and simplest to do this ourselves, instead of including a lot of unnecessary code in our project.

At first, we were exporting .obj files instead of Collada files from 3D Studio, containing our objects. However, the objects were not displayed correctly. After a lot of debugging, we figured out that the way the data was sorted in the .obj files was not compatible with OpenGL. Therefore, we decided to switch to Collada. More about this under 'after thoughts'.

3.12 PLAYER ROTATION

We wanted the players' character to always face the enemy. To do this we created a Controller called LookAtController. This controller rotates a 3d model so that it faces another model. This is achieved by using simple vector math and basic trigonometry.

When we receive the models, the positions are stored in world space coordinates. This means that the positional data originates from center of the world. In order to rotate like we wanted to, we needed both models to have their positional data relevant to the players' position, meaning player position is (0,0).

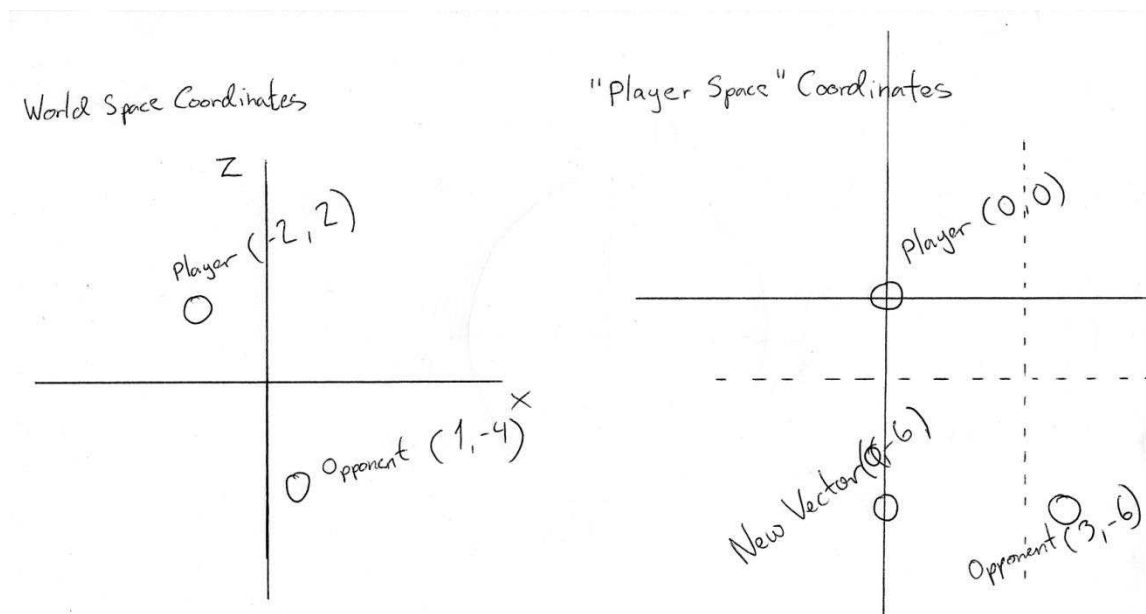


Figure 19: Changing center to player.

Deducting the player position values from the target position values, gives target values in relation to the player position. We then created a new vector where the x value is from player and the z value is from the target. Getting the dot product of these two vectors x and z components and then using the standard Java method `Math.atan(z/x)` gives us a value ranging from 0 to pi then $-\pi$ to 0.

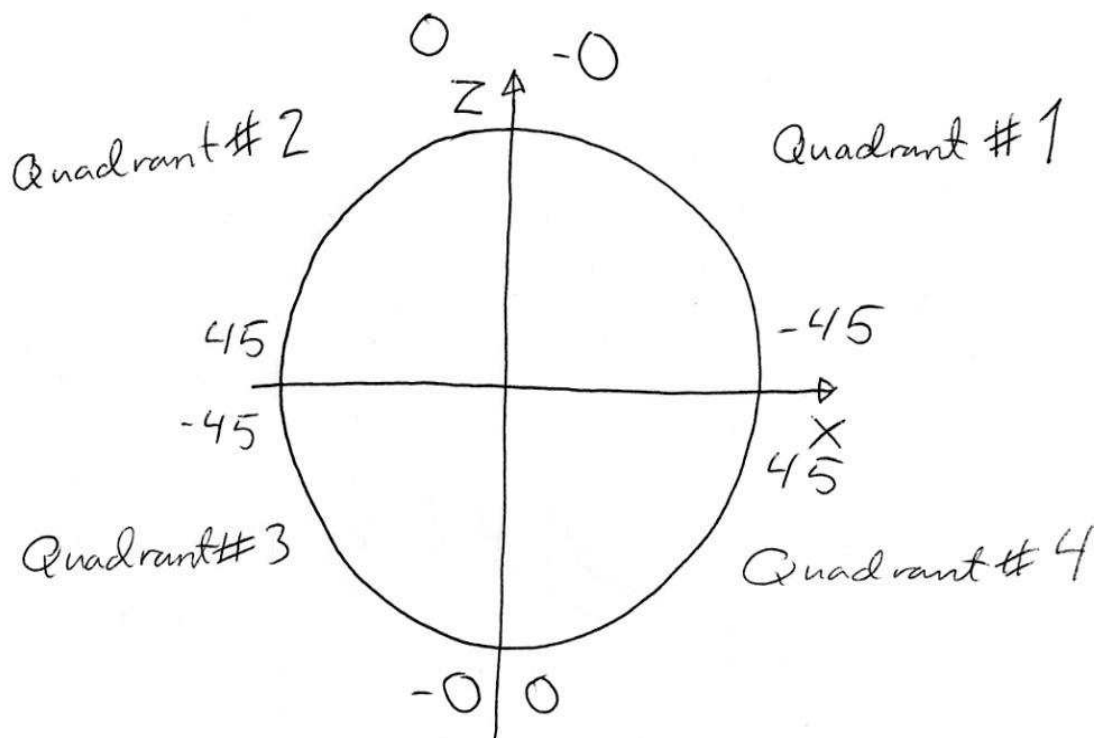


Figure 20: Quadrants and the values inside each.

We did encounter some problem when implementing this functionality: The arctan didn't return all the values we expected, but values varying from 0 to 45 degrees and -0 to -45 degrees, whereas the values returned should be from 0 to 180 and -0 to -180. We eventually decided to not track for this bug anymore and instead adjust the values according to what quadrant we were in, and the method gave correct results.

3.13 MANA REGENERATION

We regenerate the mana of the player by a small bit every second. We use a Controller called RegenController to update the players' mana through the UpdateManager loop. Not the most important feature, but it does prevent the game from stalling logically because no player has any mana left to cast spells for.

3.14 CLIENT NETWORKING

When a player moves, casts a spell, reduces his mana or health, or attacks his opponent, this has to be transmitted to the other client to hold the game in sync. We implement this by sending network packages between the two clients, containing updates to the players position, current action, etc. These packages are then read on the other client, and the appropriate actions are taken to adjust any value that was transmitted.

3.15 GAME SERVER

To allow players to find each other, and to have a “registered user”-system in place to allow for scoring boards and similar at a later time, we have implemented a game server that allows the user to register a user, log in as that user, and request to be paired with another player to play a game. So that we don't have to have a connection open to the server all the time, the client tells the server that it wants to play a game, and then checks the server at intervals to check if it has been paired. Only login and the start of a game are handled by the server. After a game has been started, all communication goes directly between the clients.

3.16 OPTIMIZATIONS

3.16.1 MEMORY ALLOCATION

We have tried in general to avoid any dynamic allocation of memory during runtime or inside member functions. Where we can avoid to, we allocate memory for variables in the class itself. If we didn't do this, loops would take considerably longer time, because the code would be interrupted by the garbage collector constantly.

3.16.2 THREADING

The network threads are most of the time blocking while either waiting for the client to create a package and queue it to be sent, or waiting for a package to be delivered to it. When it is time to stop blocking (i.e. the game is over) the threads are interrupted and gracefully shut down. With this method Like this, the threads do as little work as possible.

4 DESIGN DECISIONS

4.1 ACTION SYSTEM

The action system is our attempt at creating a manageable, versatile system for controlling the main interaction between players in our game: The spells. When starting on this part of the code, we decided that we wanted to have something that could handle anything, or at least not limit it by setting too strict rules for ourselves. This proved to be a two edged sword: On one hand you have a system that can be built upon for later projects, or create new interaction methods for this project if time allows it. On the other hand you have a system that is too generic and hard to understand. In other words: The system could have been made a lot smaller and would still perform just as good.

A system that handles player interaction will always have a higher level of complexity than many other systems, just because it is interacting with the player. As a human you expect something to behave in a certain way and you expect to get feedback when something happens. This is part of the reason why these systems get so complex.

The underlying idea of the system isn't bad, but as implementation continued and especially, when the implementation of the action system met the other systems (especially the Network system), the system started to deteriorate and started to be more rigid. In many ways, this was caused by lack of time.

4.2 UPDATE MANAGER VS. EVENT HANDLER/PROCESS MANAGER

In a game, a lot of data is being sent between systems and subsystems before anything appears on your screen. If not managed properly, the data won't arrive to its proper destination, is in risk of being exploited and can cause memory leaks. If managed too much, you will have a system that won't perform fast enough for use in any real game system.

This data also goes through many systems and subsystems, all of which perform tasks ranging from simple tasks like moving/storing data, to complex algorithms used for collision detection, physics and rendering. Newer games are also multithreaded, adding yet another layer of complexity, making it hard to understand what's going on in your code. The balance between manageable code and performance is vital in any system of a game engine, but perhaps most important in an update routine. The update routine is the system where most other systems will go to when receiving information about the game and the game world. You will always have to go through the update routine when debugging your game, so keeping it clean is important for debugging and making it fast is important for performance.

The traditional way of creating a update routine for a game is an event manager and a process manager(Not to be confused with Operating System Processes) (4). Systems that want to perform an event will send an event to the event manager, which in turn adds this event to a queue. The event manager pulls events of this queue. Each system that uses events has an event listener inside the event manager. The event manager then tells all the listeners what event just occurred and those who waits for that event will be triggered. The system receives the message that the event occurred from its event listener and performs the actions it has been set to do when this event occurs.

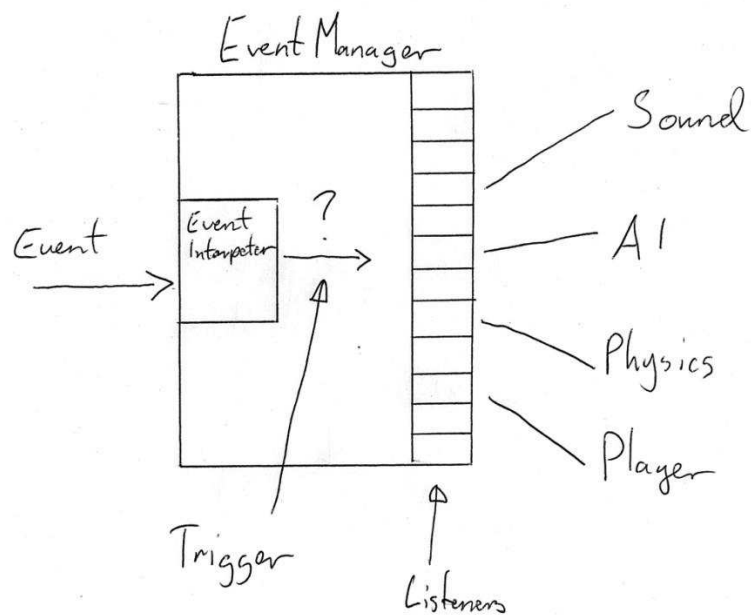


Figure 21: An Eventmanager taking a event in and sending it to the proper listener.

The difference between an event and a process is that an event is something that happened now (this tick), while a process is something that has happened and is currently ongoing. A system that just got in an event can trigger new processes.

The process manager handles processes. It adds and removes processes currently being run and it also trigger processes so they can do their jobs for the current tick.

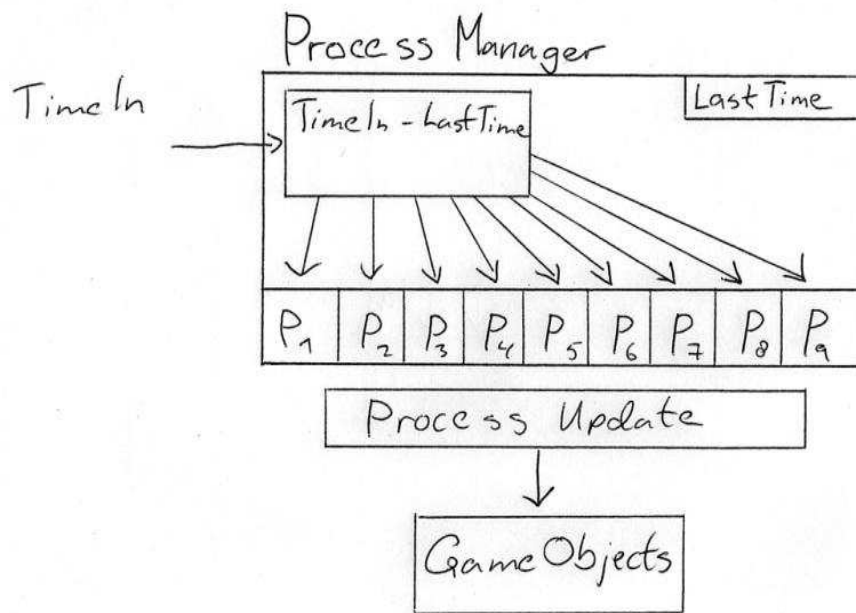


Figure 22: A ProcessManager updating its processes.

The combination event manager and process manager is a great tool for keeping large systems controlled and manageable. We decided that having two separate systems, for our project, was unnecessary. We would never reach the complexity where needing two systems, to handle something that could be done with one system, would be better. At the same time, we wanted to have a manageable system that also performed well.

Our solution was to have one manager called the update manager and several objects of classes inherited from a parent class called Controller. UpdateManager has a member function called `doUpdates()` and Controller has a `update()` function which is then overridden. The game loop runs `doUpdates()` for every frame and UpdateManager then performs all registered Controllers `update()` function.

Many of the Controllers are added to our UpdateManager inside other classes, but some Controllers were harder to find a logical placement of and were placed inside Run, where our main game loop resides. The CameraController is a good example. Even though there is only one camera in our game and it is always following the player, we chose to put it in Run to separate these. This was a design decision so that the system was less rigid.

4.3 GAMESERVER

The original plan was that the game server should handle all communication between clients, and also include a scoring board, user registration, and anti-cheat (to mention some). During this time we researched existing game servers, but most of those we found were either way too big, or for several/multi (not two) player-games. We started writing a high-throughput server in java, but after research and code writing using this plan was started, a discussion with a supervisor halted us in our tracks. Sending all data through a server when playing an online real-time-multiplayer game would create a lot of network traffic for the server, and possibly a considerable lag.

After this, communication directly between mobile units over the Internet (and over routers) was investigated. We read up on NAT, NAT-hole punching, STUN, and similar topics to have our code work between different wireless networks. At this time a proof-of-concept for sending simple packages between mobile units had been tested and found to be working.

After some time we «hit the wall», and again resorted to outside help – we were then told to ignore this problem and focus on making the code work reliably between phones, and then expand it to work over NAT/different networks if we got the time. At this point the only job of the server was to verify users, connect players that wanted to play (distribute port- and IP-settings) and saving game-scores supplied from the units. Because it was easy to combine for both server- and client side, we had the experience, and had used quite a lot longer than planned on the server, it was implemented as a REST-service using PHP, MySQL and JSON. This is the current state of the server to this day.

We could have sent binary (our earliest code did this) or XML data, but did not want the overhead of using a XML parser in our communications. We therefore settled on JSON, a simple interchange format that we easily could parse in both PHP and Java.

4.4 CLIENT NETWORKING

We quickly decided to use UDP for transporting our game data, as we felt it was simpler to implement and fit our needs. TCP was considered, but was dropped after advice. Since writing threads is quite simple in Java, and would ease the development greatly, a twin-thread network system was selected. Sending and receiving would live in to threads, but with the possibility to communicate if acknowledgement was required by any packages.

There was a discussion between us whether the network packages should be sent every xx milliseconds, or if they should be sent as soon as something wanted to update the other client with data. After the network programmer convinced the others that if they want to change it at a later time, it would be trivial, we decided to write and experience a soon-as-possible network, and change if we found any problems. To this day, the network uses this last same approach. Because of design decisions regarding the game server, client have to be on the same WLAN, or open appropriate ports in their firewalls to connect.

Because of time constrains all packages are simple strings; we did not take the time to create any form of packages-wrapper. It is however simple to replace the strings sent with more efficient packages later.

4.5 INPUT

4.5.1 MOVEMENT

Our initial idea was to use the trackball/pad/detector that is on all Android devices to control the player. Our first iterations of code used this, but we later dropped it. For one, we determined that when used, your thumb became very tired of the scroll-lift-scroll-lift movement that was required to move longer distances. On some of the devices the trackball could be used as a d-pad/joystick, while on other we had to «roll» the ball to get any movement. We also discovered that both the accuracy and possible max-speed of scrolling differed a lot between devices.

After studying other games in the same category as our own available on the Android Market, we decided to use a screen-drawn navigation icon, and parse the touch of a finger on this area into movements. Even though it was a possible solution, no form of adapter/add-on controls were ever discussed.

4.5.2 GESTURES/ACTION

There was some discussion on whether to use on-screen icons or some type of «drawing-on-screen» to start actions. Most of the games we found either had complex menus or on-screen buttons for selecting actions, but we felt this distracted the user from the game world. After research and test-implementation we decided to base out actions on screen-drawings, or “gestures”. Android had an available library that let us do this, and after overcoming some problems regarding to the joining of a game-surface and gesture-surface in Android, we had a working gesture system.

4.6 RENDERING

When deciding how the rendering was going to work, we knew we wanted something that was fast, easily maintainable and easy to debug.

The oldest method of rendering in OpenGL is called Immediate mode, in this mode you send all data from main memory to the graphics chip between every shape you draw, every frame. This is highly ineffective, but easily understood when reading the code.

The newer method using something called Vertex Buffer Objects(commonly called VBO) (5), which sends all the data to the graphics chip and stores it there until you change the actual data. This method is not supported in the version of OpenGL that we use for our project.

Vertex arrays are simpler versions of VBOs that store the data in arrays and call the graphics chip to display its data, after the data has been transferred.

This is however just about storing the data; we also need to display it. In immediate mode, we have a directly procedural method of displaying our data. We could also just store the data in a big array and run through that every frame, but this will be hard to work with inside the code. We chose to use a scene graph for holding our game data and store the data in arrays.

We wanted to also implement culling. Culling is a technique to remove objects that aren't visible from the rendering. (6) We chose to have a Boolean variable called `cullingFlag` for this. Its purpose is to mark a part of the graph which is outside of view range of the player and then mark it as culled, in a separate traverse done before the `drawTraverse()`. This will then tell the `drawTraverse()` function that anything below the node with the `cullingFlag` set to true can be skipped completely. For a larger and more detailed game world, this would be vital when attempting to achieve high enough frame rates for the game to be playable. However, it is not needed for the level of detail we have in our game world.

4.7 GAME LOOP

The game loop is a game engine's core in many ways. It re-computes the state of the game world, and handles player interaction and presents the player with the result. The two issues with game loops are the first and the last: updating and displaying the world. Both demand a lot of resources from your computer and if not balanced properly, will cause the game to slow down considerably to the point where the game is unplayable.

There are several approaches to creating a game loop. We looked at 3 possible methods:

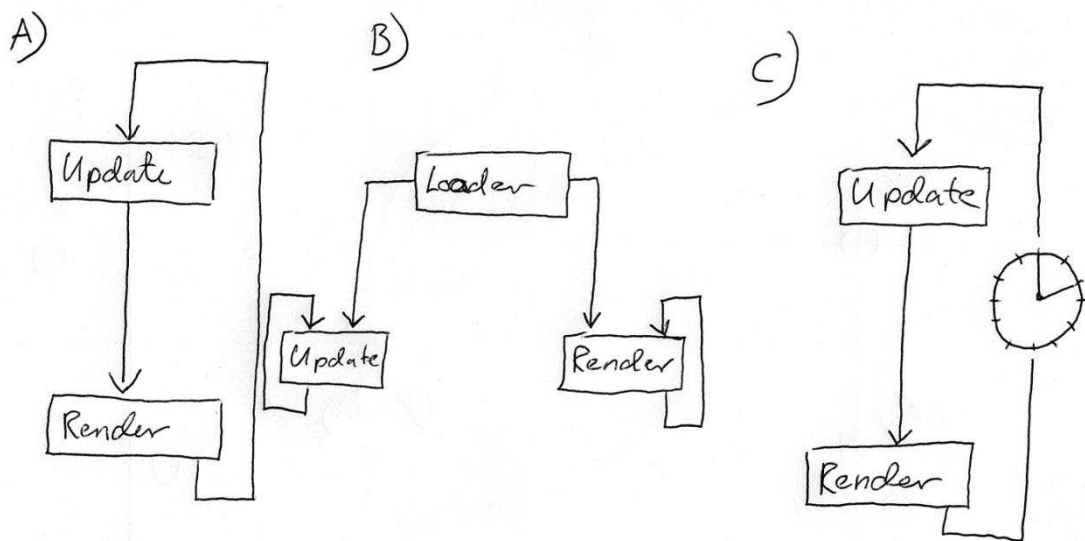


Figure 23: Methods for creating a gameloop.

In the drawing above you see 3 different methods of performing the game loop.

- A) The simplest method for creating a game loop. This method was used a lot in the early games (up to the 80s). First it does the update, then Render the result. This sounds good, but what happens when you use it on two computers that have different computational power? Take a car game for instance: On the slow computer, when your car was driving in 80 kph it would feel like 30 kph. On the fast computer however, it would feel like 250 kph.
- B) This shows a multithreaded game loop, where the update and render loop run on separate threads. Problem with a multithreaded game loop is complexity. Multithreading always introduces extra problems in terms of data sharing between

threads. If implementing a multithreaded loop, we would also need to design the game more specifically with semaphores and mutexes.

- C) Last figure is a game loop on a single thread, but uses a extra step after every run through the loop. In this step you find out how much time has been used going through the loop and then pausing for a time so that the current run through the loop matches a set frame rate you want for your game. For instance, if you want to have a constant 30 frames per second, and the run through the loop took $1/300^{\text{th}}$ of a second, you need to pause for $9/300$ seconds to achieve 30 fps.

We chose to implement C as it would give us a stable game loop that would hold a stable fps, without need for complex methods using semaphores and mutexes.

4.8 COLLISION

There are many ways of representing an object in a collision system, called bounding shapes. What they all have in common is that they need to encapsulate the object completely. At the same time, they should not extend the boundaries of the object more than necessary. This can be done using spheres or cubes, or with two-dimensionally representations like circles or squares. The more accurate the bounding shape is, the slower the collision system gets.

Since we don't need to check for collisions above the ground level, we don't need 3D shapes. We could have used bounding circles, but then we would still have to use squares for objects like a rectangular wall. Since we wanted to stick to one type of shape, we decided to use two-dimensional squares/boxes.

For the static objects, the bounding box is stored in the geometry node. For the two player objects, the bounding box is stored in the translate node instead. The reason why is that since the players are dynamic objects, the bounding boxes have to follow their objects when these moves. Because of this, it was most convenient to store the bounding box in the

translate node. When the translate controller is connected to a translate node, the bounding box gets copied to the controller as well, and then added to the grid in the collision system. There are more logical classes to store bounding box, but this would be problematic when checking for collisions. For instance, if we were to store this in the Human class, we would have to take the bounding box in as parameter from the Update Manager. This solution is inflexible and would cause problems if we wanted many bounding boxes.

5 IMPLEMENTATION

5.1 NAMING CONVENTION

We use camel case as naming convention for our functions and variables.

Examples:

```
thisIsAExample();
```

```
thisIsAVariable = 0;
```

5.2 ORGANIZATION OF THE CODE

The project consists of three parts:

- 1) The game itself, with user registration, login and menus
- 2) The game server, a web server written in PHP
- 3) A Java program that reads in game object models and creates the initial scene graph file.

Since our game is written in Java, the source code is organized in packages. We have one package for each logical part of the game.

The packages are:

Actions

This is where the system for casting spells is located.

Collision

Contains the collision manager, with a grid of cells dividing up the world.

Controllers

The controllers, such as translate, rotate and animation, that controls and changes their respective nodes in the scene graph.

DataStructures

We have a few own data structures here, such as vectors and bounding box.

Exceptions

Our own exceptions

GUI

This GUI is the Heads Up Display (HUD), which includes the steering wheel for the player, and all the information shown on the screen during gaming.

Human

This is where we represent the two players, both our own player and the opponent (network player).

MainMenu

This is where the GUI with user registration, login and main menu is placed.

Network

Contains the networks sender and receiver. This is where the network communication is handled.

Run

Here we have a run class, which is the first to run when the game starts. The game loop is located here as well.

SceneGraph

The scene graph code is located here, with all of the different types of nodes in the scene graph.

UpdateManager

Contains the update manager, which keeps track of the controllers update routines.

Utils

Utilities, like constant values. Used partially for testing purposes.

5.3 DESIGN PATTERNS

5.3.1 SINGLETON

This design pattern results in a class that there can only be one instance of. Meaning you can call it from anywhere in your code. This makes manipulating the class very easy.

In an application, you want a manager class to contain specific nodes from a graph. In order to do this, you need to traverse the graph and identify the nodes that you want and then add them to the manager class. One way to do this is to take the manager class in as argument into the traversing function of the graph. You will then have access to the manager class inside the node. However, what if the traversing function already has 3 arguments? And what if you need to do this for more manager classes? Suddenly you will have a large parameter list for the function that is getting unreadable for outsiders. On top of that, the function now needs 3 manager classes as arguments, making the graph function very rigid and hard to call anywhere else but specific locations in the code, where you have all 3 manager classes available.

With the manager class as a singleton, you no longer need to worry about parameter lists for the graph function, but instead ask for an instance of the manager class inside the function. This also scales well with having more manager classes; you can call them inside the function where you need them.

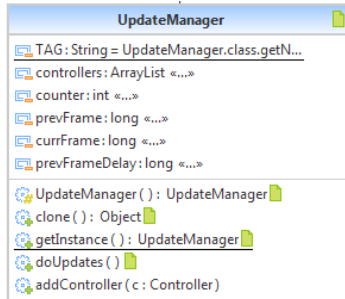
To implement a singleton in java you need to do the following:

1. You need a static variable of the class itself set to protected or private.
2. The constructor of the class needs to be protected or private.
3. You have a public member function (usually) called `getInstance()`.
4. You need to disable object cloning.

The programmer then calls the class by using `getInstance()`. This function then checks if the static variable is null. If yes, then create a new object and assign the static variable to it. If no, do nothing. After doing this check, return the static variable.

5.4 NO.HIG.RAG.UPDATEMANAGER

5.4.1 UPDATEMANAGER



The UpdateManager has two main purposes: Adding new controllers and traversing them during the game loop.

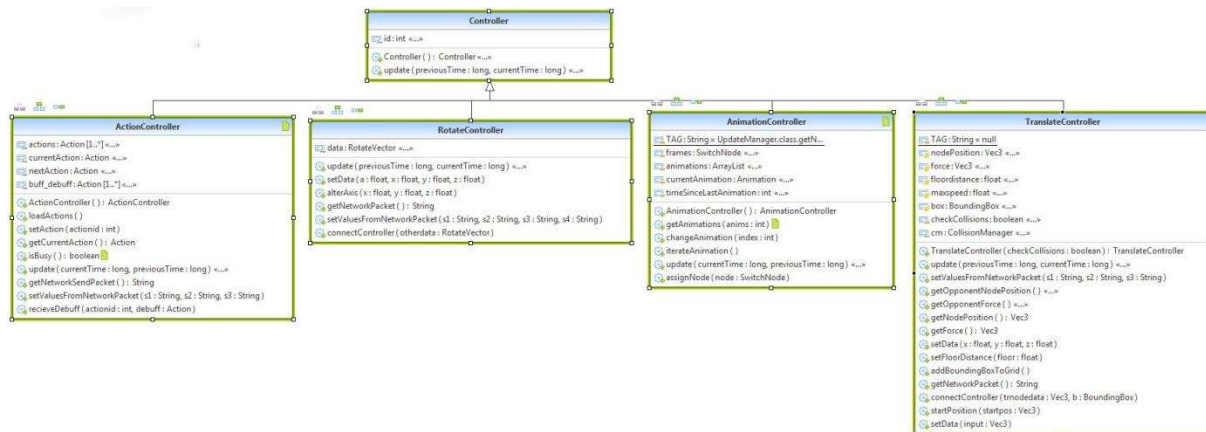
The UpdateManager contains a `ArrayList<Controller>` which stores all controllers for the `doUpdates()` function to iterate through and call each Controller's `update()` function.

The UpdateManager is created as a singleton.

Besides the functions made for the singleton design pattern, the UpdateManager is a very compact class, only having two functions, one for adding controllers and one for the update routine(`doUpdates()`).

Before doing updates, the UpdateManager gets the current time in milliseconds and then passes current time and previous time into each controller's update function. After iterating through all the controllers, it stores the current time as previous time.

5.5 NO.HIG.RAG.CONTROLLERS



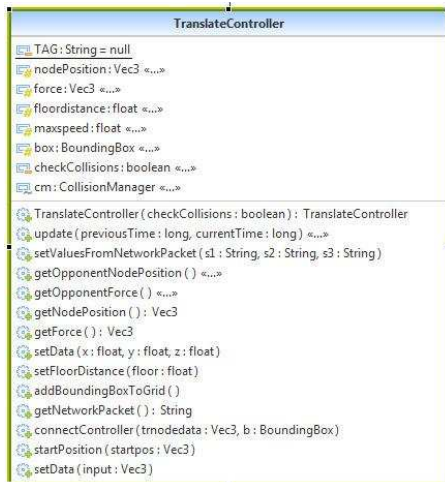
5.5.1 CONTROLLER



Controller is the parent class. Its purpose is to allow inherited classes to be used in the UpdateManager, making sure that the functions UpdateManager needs from Controllers are present. It contains a id value for debugging purposes, a constructor and a update() function. The update() function takes two arguments when called; currentTime and previousTime, these variables are used in certain Controllers to measure its effect set in the timespan that has occurred since the last round of calculations. Each child class of the Controller overrides the update() function with its own.

The TranslateController and RotateController both connect to Nodes in the SceneGraph. By doing so, these controllers can manipulate the nodes without ever being inside the scenegraph. This connection is done in the Controllers constructor.

5.5.2 TRANSLATECONTROLLER



Moves an object around in the world based on current position and the force that the engine asks to move the object with. This force can be of any size, so it is the controllers' responsibility to ensure that the move is legal. Illegal moves can be moves that end up inside another object or moving faster than what the object is allowed to move. The latter is prevented by having a

variable, here called 'maxspeed', which will cut off the force vectors length before applying the new position.

The CollisionManager handles any objects colliding and alter the force vector to the proper position based on object collision.

The TranslateController is currently used on two dynamic world objects; 'Player' and 'NetworkPlayer'. We didn't want either objects to have separate controllers for input or network handling of data, so we created one class that would support both ways and added extra functions for handling network packages as input and output. We use a flag in the constructor to tell the controller if it is being used for network or player input, 'isOwnPlayer', which turns off collision checking for the NetworkPlayer. We do this because the network packet has already been checked for collisions, so the controller should not check again. This does raise security issues, but a better solution is beyond the scope of this project.

setValuesFromNetworkPacket() takes 3 strings in as parameter and assigns these values to the force vectors x, y, z values, allowing the update routine to run as normal, without ever knowing if the input was from gestures or from network packages.

getNetworkPacket() will return a packet to the network system to send, which contains the force vectors x, y, z values.

The players' bounding boxes, used by the collision system, is stored in the translate controller. This is because it's used to check for collisions between the moving object (a player) and other objects in the world.

setData() has two versions, one using a Vec3 as parameter and one using 3 floats for x, y and z. Both do the same; setting force x, y and z values. Some classes use Vec3 objects for storing positional data. While in streams of data, like in a network or file loading system, the data appear as raw data, not objects.

The update() function from Controller is overloaded and created to do the translation for this tick.

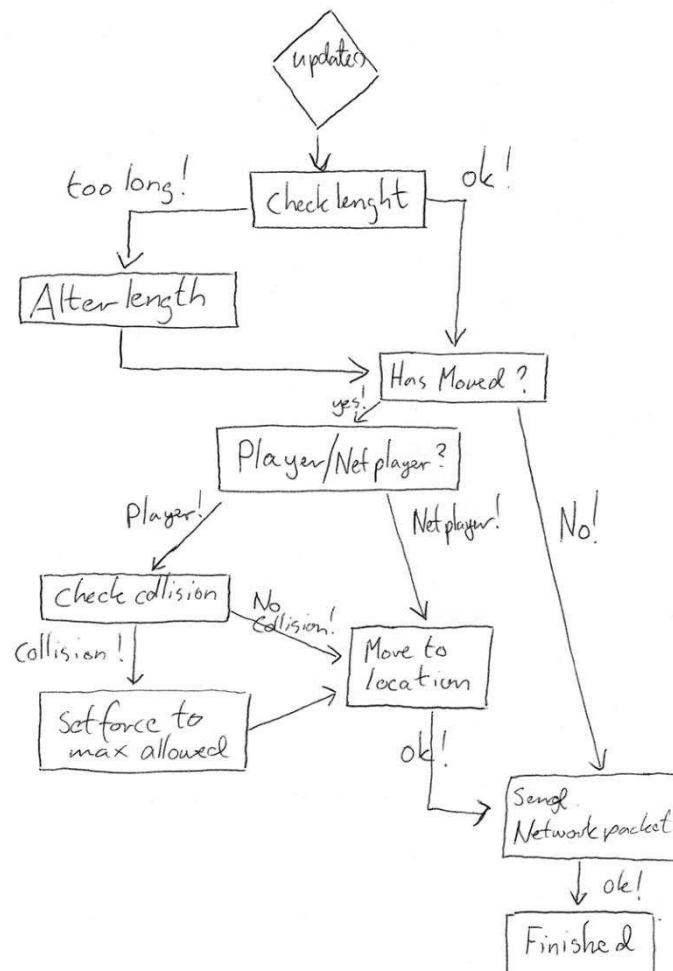
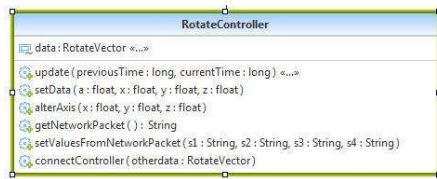


Figure 24: How a TranslateController is progressed.

5.5.3 ROTATECONTROLLER



Rotate controller manages rotation of an object. The Controller connects to a RotateNode from its

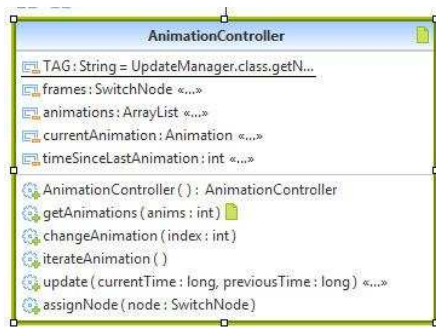
constructor. This means the node values will be the same as the controller, meaning any values updated in the controller, will be updated for the node at the same time.

The Controller has two variables of the type RotateVector. One is used for the values you *want* to rotate by, and the other is the values that will be set for both controller and node. Since we want time dependent rotation and not frame dependent rotation, we need to take time into consideration(provided by UpdateManger). We multiply the angle we want, by the time elapsed since last frame (converted to seconds) and save this in the shared variable, giving us frame independent rotation.

The functions setData() and alterAxis() changes the values of the non-shared variable(the one we *want* to rotate by).

As any controller, the RotateController has functions for returning its values & modifying its current values by a networkPacket.

5.5.4 ANIMATIONCONTROLLER



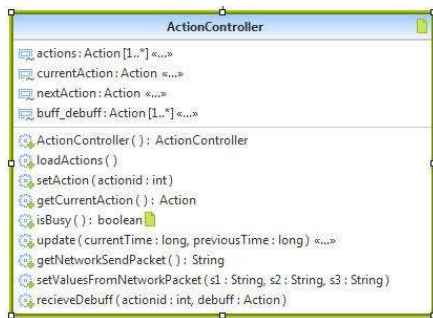
The AnimationController controls what animation and what frame is being displayed on screen. It is connected to a SwitchNode and this node is given commands about what frame to display. The AnimationController contains a nested class called Animation. We chose to

create this class inside the AnimationController because there is no relevant use for it anywhere else. The Animation class contains 3 variables: start, end, current. These variables refer to the indices inside the SwitchNode that relate to the start of the animation, the end of the animation and the current frame being rendered.

Apart from the functions used while the game is running, there are also functions for starting up the game; Functions for loading the animation list, assigning a switchnode to the controller, functions for the Player to iterate an animation inside the Controller.

We also check the time taken since the last time a frame was displayed. This is to have a internal variable to check against when running the gameloop. This ensures that the animation is played smoothly and not uneven.

5.5.5 ACTIONCONTROLLER



The ActionController controls what action is currently in use and the action itself manages what to do in time space when an action is being performed. To avoid having two separate controllers for Player and NetworkPlayer, we designed the controller so that it can

take commands from networkpackets and from gestures. As with any Controller, its main functionality lies in the update() function. This function first checks if the actionController currently busy(currentAction != null). If not, it executes currentAction.performAction() who takes the current time and previous time in as arguments as usual. We also use a NullPointerException when using performAction().

If the ActionController isn't busy, update() also checks if currentAction is finished. When finished, currentAction will be reset and be set to null. This tells the ActionController that it is currently idle.

For networkpacket send/receive the functions getNetworkSendPacket() and setValuesFromNetworkPacket() are used. The first function returns a string containing the current actions values. The latter function receives the package and stores the data into the current action.

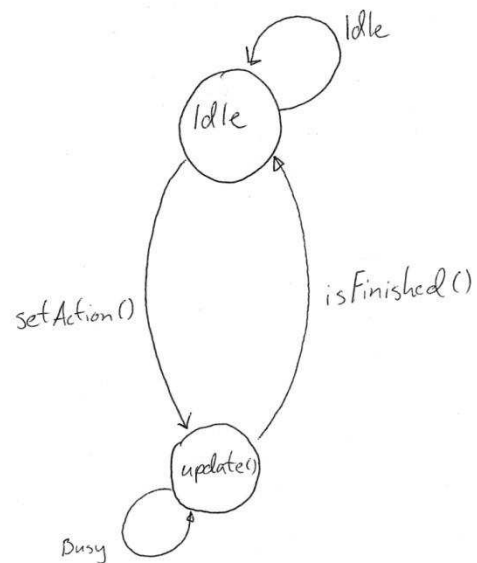


Figure 25: States in the ActionController

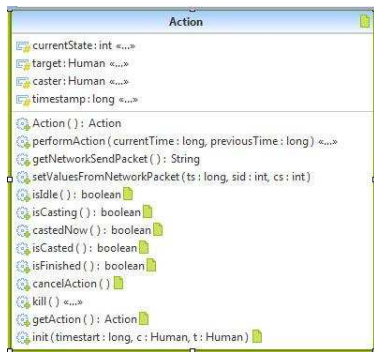
5.5.6 REGENCONTROLLER

This is a controller for status that should regenerate over time. In our implementation it only regenerates mana at a set rate, but can be extended to regenerate health, degererate either, or simply increse and decrease the rate.

5.6 NO.HIG.RAG.ACTIONS



5.6.1 ACTION



The Action class is designed to be a parent class that holds the basic functions used by the ActionController. Its functions are those designed to manage the creation, update and logic of an action. It does not hold any specific routines as to how an actual action is going to be used, but functions that help

decide if it *can* be used.

Action also has functions for saving and loading files to/from networkpackets. For saving the data, we store it into a string and for loading the string from the networkpacket has already been interpreted into the correct variable type.

The ActionController uses the performAction() function inside the Action class. Any inherited class needs to override this to fit its own needs.

Init() is a function that in the current system isn't used, but in a more advanced Action system would be essential(See discussion about the Action System).

An Action has 4 important variables: currentState, target, caster, timestamp.

currentState:

this variable is an int, that we use to store an actions 5 different states: 0 means the action is idle, 1 means it is currently being casted, 2 says that it was casted *now*, 3 says it has been casted and 4 says its finished. When reaching state 4, it means it shall be set to 0.

Target:

Says what Human object the Action was meant for.

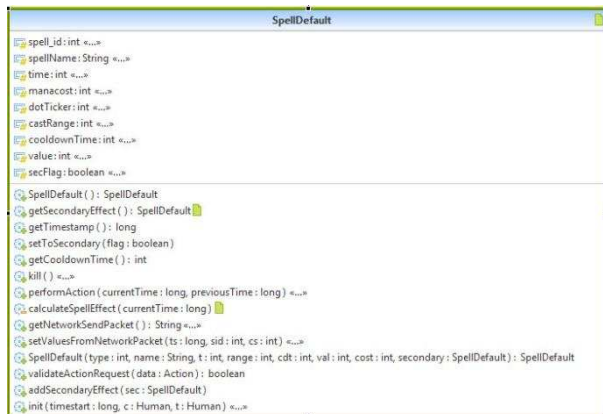
Caster:

What Human object the Action was originated from.

Timestamp:

The timetick when the Action was started.

5.6.2 SPELLDEFAULT



The SpellDefault is a class which handles the internals of how a spell is going to work. It has been made as a generic spell class, that is meant to handle both damage and healing to a target and both directly and over time. It is a child of the Action class and therefore

inherits its variables and functions. In addition it has extra variables and functions for debugging and game logic/game play. SpellDefault is designed as a Finite State Machine (FSM) and uses 3 member functions to define the body of code used in the FSMs states and transitions.

Variables:

Spell_id and SpellName are metadata variables used for identifying the spell both in debugging and in the game.

The variables time, manacost, dotTicker, castRange, cooldownTime and value are all used for defining the spells behavior. If the value is a negative number, it will deduct health from the target; if its positive it will heal it. dotTicker is used instead of value if it is an over-time spell, meaning it increases or decreases health of the target slowly over time.

castRange and cooldownTime both specify rules regarding the spell being casted. castRange says maxrange the caster can be from the target and cooldownTime is the time in milliseconds that the caster has to wait before he can cast the spell *again*, after having casted the spell already.

Member functions:

SpellDefault splits its main functionality into 4 functions, performAction(), init(), calculateSpellEffect() and kill(). We chose to do it like this because of readability.

PerformAction() controls the states of a spell.

init() sets up the spell so it's ready for calculation.

calculateSpellEffect() calculate effect based on spell_id and input from the system(like time).

Kill() resets the SpellDefault(Called from ActionController).

5.7 NO.HIG.RAG.GUI

Since all touches are captured by the main activity (no.hig.rag.run.Run) this file contains code to initialize and pass-through information. The flow of touches is Finger → Run → Gestures → Movement.

5.7.1 GESTURES

All touches on the screen are sent to a GestureOverlayView. This View is connected to the android.gesture-library. All incoming touch-events on the screen surface is automatically compared to prerecorded gestures, and an event is fired from the library when a match is found. In the initialization of the gesture library we set no.hig.rag.gui.TouchEventParser as the target for a copy of all touch events – this way we can use the screen for both gestures and movement.

When a event is fired the value of the gestures score (how close it was to the original gesture) is compared to a constant. We found that a score between 1.7 and 2.5 were the outer limits of “really bad match” and “pretty good match”. If the gesture passes the score-check the gesture is converted from a string (as represented from the gesture library) to a constant and passed along to the no.hig.rag.Controllers.ActionController.

5.7.2 MOVEMENT

All touch events are passed from the gesture-library through to no.hig.rag.gui.TouchEventParser. We are only interested in touches that are within the area of the rendered navigation icon. For this purpose the HUD updates the TouchEventParser (a singleton) every time the position of the navigation icon changes (I.e the screen rotates).

When we get a touch, the coordinates (x,y) are compared to the latest values we got from the HUD. If these are outside our area, we discard the events. If they are within x-min-x-max and y-min-y-max we translate the so they are relative to origo in the center of the navigation

icon. The length of the vector from the center to the point is calculated, and if they are outside the circle of the navigation icon, they are discarded.

Once a set of coordinates has passed these testes, they are translated to screen-size-independent values. The length of the vector is converted to a power-value between 0-0.2 and 1. Values below 0.2 are set to 0 to make it easier to stop the movement by moving the finger to the navigation center. The coordinates are also converted to the arc tangent of y/x within the range $[-\pi..pi]$.

The power (0-1) and arc tangent ($-\pi$ - π) is passed on to `no.hig.rag.Human.Player`, which will use these values as a force to move when next rendered.

5.8 NO.HIG.RAG.NETWORK

5.8.1 NETWORKSENDER

It is initialized in `no.hig.rag.Run.Run` to send packages to a target IP and port. Once the thread has started, it attempts to take() a package from the blocking list `networkPackets` in `no.hig.rag.Human.Player`. The thread will block/wait/sleep here until there are any packages to be sent. Once Player has created a package, the thread continues, and transmits the new package. It the returns to take() and waits for the next packet.

5.8.2 NETWORKRECIVER

The `NetworkReciver` is much like `NetworkSender` – it is initialized in `Run` to listens to the default port (9876). It blocks on the `recv`-method of a UDP socket. Once a package (String) is received, it is either sent to the `Player` or `NetworkPlayer`, depending on the content in the packages.

5.8.3 COMBINED

Both sender and receiver are Java threads, running in their own processes.

Because the threads are blocking most of the time, they are stopped by issuing a interrupt.

The code is expecting this, and shuts down gracefully.

The packages are sent via UDP for highest possible speed. This does not guarantee delivery, but most of our packages send data that does not need acknowledgement.

Because of time constrains, all packages are simple strings.

5.9 NO.HIG.RAG.COLLISION

5.9.1 COLLISIONMANAGER

The collision manager is class with a singleton object that manages all the collisions.

The class holds the grid of cells dividing up the world. Each cell has a list of the objects they hold at the time, and each object knows which cell(s) it is currently in. This way, we only need to check for collisions between objects that are in the same cell(s). This grid is represented as the two-dimensional array of Cell-objects, cells, stored in the CollisionManager.

CheckCollisions() is used to check one object for collisions. It finds out which cells this object is in, and calls all of these cells own checkCollisions().

5.9.2 CELL

CheckCollisions() gets the object to check against as a parameter, and checks this by calling checkOneCollision() for every object the cell holds at the moment. It is in

CheckOneCollision() the actual collision check is happening. We check if any of the bounding boxes' edges overlap. If so, handleCollision() is called. HandleCollision() first finds if this was a collision in x-, or z-direction. It then calculates the angle of the move in relation to the other object. If the move is more than 45 degrees away from the object we collide with, the object will move the desired distance in the direction parallel to the edge of the bounding box it is colliding with, but it will not move in the direction towards the object. From this, the bounding box is either moved back or moved along the edge of the other object.

5.10 NO.HIG.RAG.HUMAN

We have two wizards (players) in the game. These are each represented with their own branch of nodes in the scene graph. This makes sure that the players are drawn to screen. However, to be able to play the game using the players, we need some kind of logical representation of the players as well. The players' information, like health, mana and movement need to be linked to each of the players on the screen. Further, the player and network player need a link to the user input and network input, respectively.

We thought the best way of doing this was to have a singleton object for each of the two players, where their respective classes inherit from the same parent class. This way we can connect to the players from anywhere in the code.

5.10.1 HUMAN

Human is the parent class for Player and NetworkPlayer. This is used for all of the properties that the player and network player share.

Human stores all the information about the players, like the players' health and mana. It also holds the controllers needed to control the players. This include an action controller for casting spells, a translate controller for moving the player and two rotate controllers for rotating the upper and lower body. It also includes two animation controllers for switching between the different animations of the players upper and lower body, in the different stages (idle, walking or casting a spell).

When we want to change something related to the player, we get the players singleton object. Human (its parent) then has methods for either modifying the nodes in the scene graph (through the controllers), or modify the players properties directly. For example, `setMove()` uses the players translate controller to modify the players position in its translate node, while `alterMana()` modifies the mana directly (since mana is stored in Human).

ConnectControllers() gets/finds the nodes representing the player(s) in the scene graph, and assigns these nodes to their respective controllers. It also adds the controllers to the update manager, making sure that the controller's update routines get executed.

RecieveAction() gets an action id, and, if the player is not currently in an action (casting a spell), sets the new action as the current one.

5.10.2 PLAYER

The singleton object of the class Player is representing "our own player" in the game (as opposed to the opponent's player). Inherits from the base class Human.

The players position is updated through updateMovement(). This method gets information about a touch on the navigation icon on the screen, and converts this to a move in the correct direction. It then sends the move coordinates to the translate controller.

CreateNetworkPacket() builds up a network package that contains all the current information about the player. To do this, it uses the controllers getNetworkPacket() methods, to get the current information from these as well. This package will be transferred over the network to the opponent, so our player gets updated on the opponents phone as well. The package is a string of values in a defined order, separated by commas.

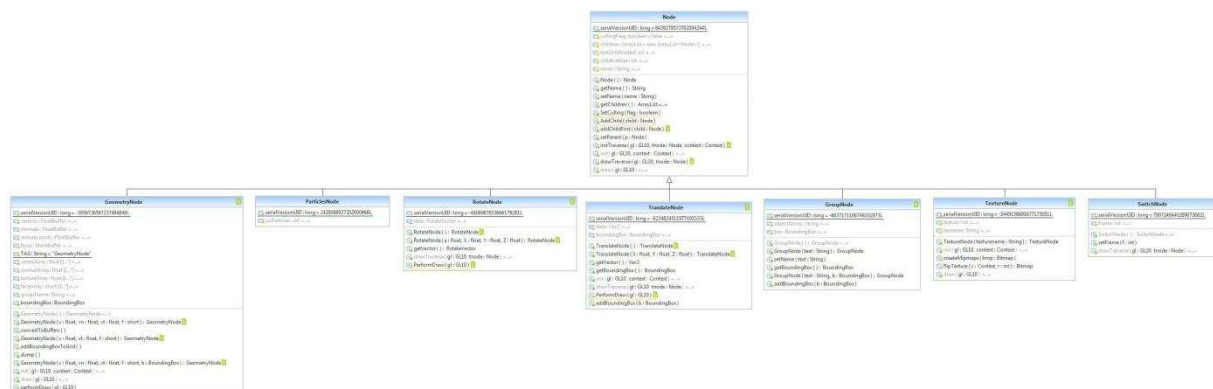
5.10.3 NETWORKPLAYER

The singleton object of the class NetworkPlayer is representing the opponent's player (the network player) in the game. It also inherits from the base class Human.

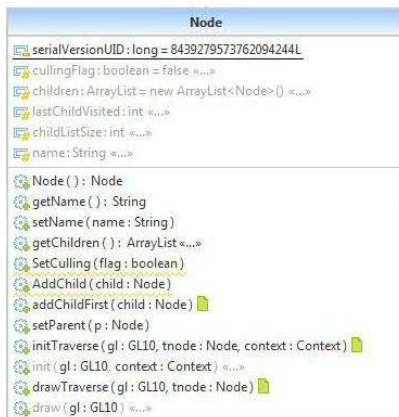
SetValuesFromNetworkPacket() gets a network package as a string, containing all the current information needed about the opponents player. It then extracts the different values from the package, and assigns them to the right properties of the network player. It also gives the controllers their updated information.

5.11.1 SCENEGRAPH

5.11.2 NODES



5.11.3 NODE



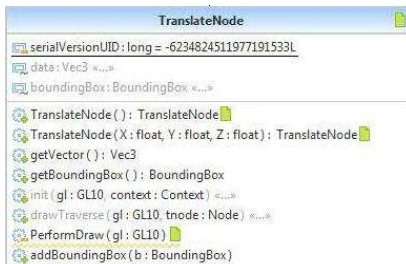
This is the parent class; it contains the variables serialVersionUID, parent, cullingFlag, children, lastChildVisited, childListSize and name.

The important variable here is 'children'. 'children' is a ArrayList that contains all the node's children nodes.

The variable parent is a link to the nodes parent node, as of now this is not used, but it is a valuable variable to have for debugging purposes (Being able to see both up and down a scene graph when looking for errors).

The parent class Node also contains most of the basic member functions that are involved when building up the scene graph. Functions for adding a child, setting the parent, setting the cullingFlag(), setting/returning the name of the node and functions involving the initialization of the nodes(More on this later).

5.11.4 TRANSLATE NODE



The TranslateNode moves its children nodes around in world space. The positional data is stored in a Vec3 object, and used for translating its children nodes to the position stored. The drawTraverse() calls performDraw(). The

performDraw() functions purpose is to move to a specific location and use this new location as center for any child operations. The performDraw() achieves this by using the OpenGL call glTranslatef().

5.11.5 ROTATE NODE



The RotateNode, rotates its children objects around the current center point. Just like the TranslateNode, it first performs `glPushMatrix()`, does its rotation, calls all its childrens `performDraw()` and then performs `glPopMatrix()`. The rotational data is stored in a class called RotateVector. This class contains an angle variable and 3 orientational variables(x, y, z) that define over what axis the rotation is to be performed on.

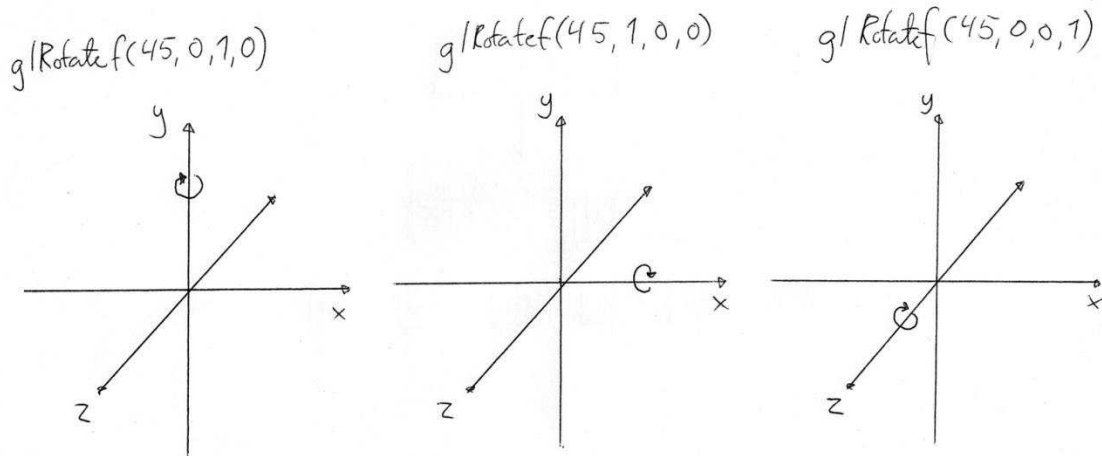


Figure 26: illustrates the axis settings and how they affect rotation.

5.11.6 GEOMETRYNODE



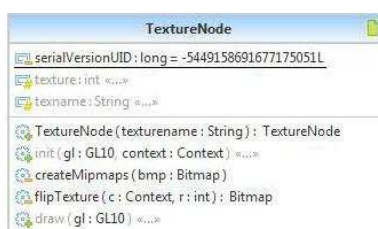
The GeometryNode is the container for the data used in OpenGL to draw a 3-dimensional object to the screen. It holds the buffers for the vertices, normal and texture coordinates. It also holds the bounding box for the collision detection system.

The file loading system also uses the

GeometryNode to convert data from file to proper data values that are fast and easy to read for OpenGL. We chose to have these functions inside the GeometryNode for the ease of use.

When first initialized, the GeometryNode stores the incoming arrays of data, into its own arrays. After being initialized, it then converts these arrays into float buffers and short buffers. Doing it this way, allows us to load the files much faster than reading and converting the files into buffers in runtime, due to serialization in Java.

5.11.7 TEXTURENODE

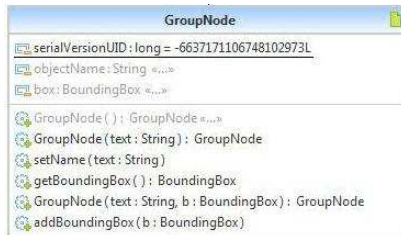


The TextureNode sets up a texture to be used for its children nodes. The performDraw() itself is just a OpenGL function that lets OpenGL know what texture to use. OpenGL stores

textures in memory and returns a handle to your code. If you want to use this texture you need to enable this state and let OpenGL know what texture you want to use. By storing the handle when you load the texture, you can then easily use the texture later on. The texture data is not needed in your application after loading successfully into OpenGL, so it can be deleted afterwards.

One difference from Windows/Linux and Android is that Android stores the texture differently, resulting in a flipped texture compared to what is normal in OpenGL. This was solved by having a function that flips the texture before we send it to OpenGL.

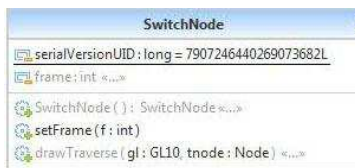
5.11.8 GROUPNODE



The GroupNode is a metadata node used for describing nodes below it. Its main purpose is for debugging. A scene graph can quickly become very large, even for small worlds.

The Player class alone uses 6 nodes, not counting any GroupNodes or GeometryNodes used by the AnimationController. The number of nodes quickly rises to a level where it becomes hard to determine where you are in the graph. Having GroupNodes helps the programmer navigating in the code while debugging.

5.11.9 SWITCHNODE



The SwitchNode takes advantage of the children arraylist and uses it differently from the other nodes. Instead of traversing all its children, the traversing will visit *one* node. This node is set in the setFrame() function. SwitchNode is used in the AnimationController. Each child in the SwitchNode refers to a GeometryNode. Every GeometryNode is then one frame in one or several animations. Using the setFrame() function you can easily control what object is being displayed.

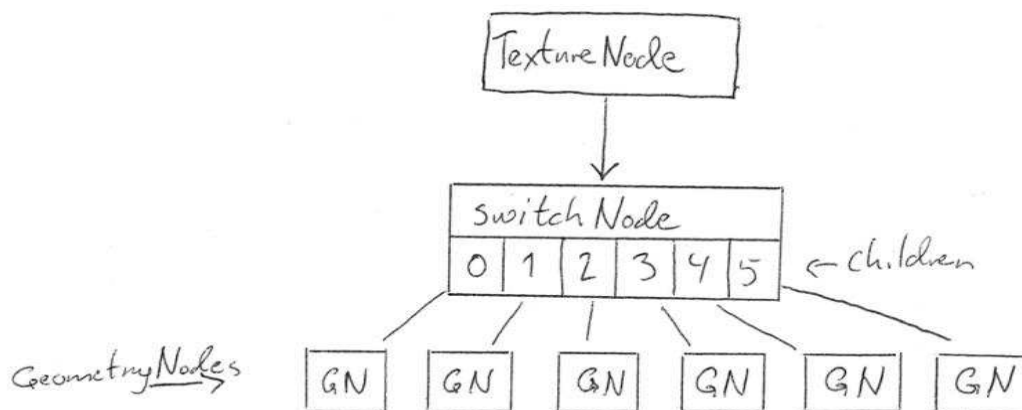


Figure27: SwitchNode uses its children list to set which child to draw, instead of drawing all children

5.11.10 INITIALIZATION OF NODES

The GeometryNode and the TextureNode are the nodes that need initialization before being able to be drawn. This is due to serialization. The data used by OpenGL is stored in buffer object, while the data loaded is stored temporarily in arrays.

After the scene graph has been built, it needs to be traversed once to initialize the nodes. This converts the data so it can be used by OpenGL.

After the scene graph has been built, it needs to be traversed once to initialize some of the nodes. The GeometryNode and the TextureNode need initialization before they can be drawn. InitTraverse() in Node traverses all the nodes once, and runs the nodes' init(). Init() is empty in all the nodes except GeometryNode and TextureNode, where it is overridden. The initialization only happens once.

GeometryNode holds the vertices-, normals- and texture-coordinates, and face indices. The node needs an initialization because of the format of these data when written to file, in contrast to the format needed by OpenGL. The data used by OpenGL is stored in byte buffers, while the data loaded from file is stored temporarily in arrays. ConvertToBuffers() converts the arrays into byte buffers of the wanted type (float or short). The reason why we store all of the geometry coordinates temporarily in arrays instead of using byte buffers right away, is that byte buffers are not serializable, so we have no way of writing them to file. More on serialization in the next section.

For static objects, the objects' bounding box is stored in the geometry node. For these objects, init() adds their bounding box to the collision grid as well.

TextureNode needs initialization for loading of the objects' texture, from the resource folder of the project.

5.11.11 SERIALIZATION

Serialization of an object is the process of converting the object into a sequence of bytes. These bytes can then be written to a file, for example. Deserialization is the opposite, (7) i.e. to convert the bytes back to an object. In Java, for an object to be serializable, and thus be able to be written to file, its class (or a class it's extending) has to implement the interface `Serializable`.

Every serializable class needs a unique version number, called `serialVersionUID`. This is used to compare the class when it was written to file, and when it is loaded back from file, to see if the two class versions are compatible. If the `serialVersionUID`'s don't match, it will result in an `InvalidClassException`.

As discussed earlier, we have a separate project called `CreateSG`, which creates a file with the initial scene graph. This project is connected to the game project, so it uses the same classes (in the package `SceneGraph`) as the game, when building the scene graph. The classes `SceneGraph` and `Node` (which is the parent of all the nodes) implement the interface `Serializable`. In addition, all of the nodes have its own `serialVersionUID`.

5.12 NO.HIG.RAG.DATASTRUCTURES

We have made some simple data structures, helping us to store data in a simple and efficient way.

5.12.1 VEC3

A simple class holding the three float variables X, Y and Z, which are coordinates in the worlds' coordinate system. The class is used throughout the project, in places related to movement/translation of objects. The three variables are made public. Even though this is bad object oriented practice, it reduces the number of method calls, and increases the performance.

5.12.2 VEC2

A simple class holding the two float variables X and Z, which are coordinates in the worlds' coordinate system. Used in the collision system, to compare the position of objects to each other. We don't need to check for collisions along the Y axis, so we don't need the Y coordinate.

5.12.3 ROTATEVECTOR

A simple class holding the three float variables X, Y and Z, which are coordinates in the worlds' coordinate system. The class also includes an angle. This is used in rotate node, for rotation of objects. The angle says how many degrees the object should rotate, while the coordinates say in what direction it should rotate.

5.12.4 VECCELL

A simple class holding two integer variables, X and Z. These are used to represent a cell in the grid dividing the world, in the collision system.

5.12.5 BOUNDINGBOX

Represents the bounding box that encapsulates the objects. Used for collision detection. The class holds the box's coordinates in the world, along with its own local coordinates. It also knows which cells in the grid the object is currently in.

5.13 TOOLS

5.13.1 PARSING GAME OBJECTS AND CREATING SCENE GRAPH FILE

The code for the project that parses the Collada files containing the game objects, and creating the file with the initial scene graph, is described below.

5.13.2 MAIN

Here we set the name of the Collada file containing the objects, and the name of the scene graph file. We then make a new BuildInitialSceneGraph object. (8)

5.13.3 BUILDINITIALSCENEGRAPH

This is where the initial scene graph is made. We make an instance of ColladaParse, which returns a list of GameObject objects. This list contains all of the game objects from the Collada file. Then the two players are added to the scene graph. The players' geometries are identical, but they have different textures. Each player has its own method for getting added to the scene graph. This was considered the most convenient way of adding them. There is a separate method for adding the static objects to the scene graph. When adding an object to the scene graph, we add a new branch under either the dynamicNode (players) or the staticNode (other objects). Then the branch is built up by the nodes needed.

5.13.4 COLLADAPARSE

The Colla file gets parsed, and for every game object that is found, a GameObject object is added to an array list called gameObjects. GetObjectList() returns the list of game objects.

5.13.5 GAMEOBJECT

A class for holding all the information needed about a game object. This class is only used temporarily, until the objects are written to the scene graph file. MakeBoundingBox() finds the objects maximum and minimum coordinates, and makes the objects bounding box.

5.14 SERVER

The server is implemented using PHP, MySQL and JSON. It is primarily 6 PHP-files, each with their own function. The clients send data using HTTP POST with data formatted in JSON, and the server responds with a body containing a JSON-formatted response.

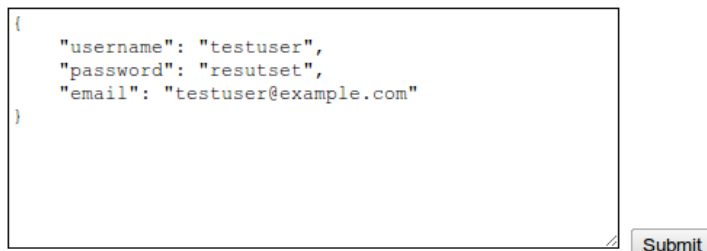
5.14.1 COMMUNICATION

Our client-server model is based around JSON-requests and replies. They are sent as normal HTTP POST requests, but only contain JSON formatted data.

An example follows for registration of a new user:

The client sends a username, password and an email to the server (illustrated from the server-test files)

```
{
  "username": "testuser",
  "password": "resetset",
  "email": "testuser@example.com"
}
```



If the server accepts the data sent by the client, the username is free, and the email is correct (it is a valid email) a new user is created. The server returns a JSON-object with the success-field set to true, the username and e-mail used when registering, and a session-value that must be used on subsequent requests to identify as this username.

```
{
  "success": true,
  "username": "testuser",
  "email": "testuser@example.com",
  "session": "ea5c06434e5a483539e45f40947a2c6d",
  "message": "Login successful"
}
```

If the username is busy, something was wrong with the data, or there was a technical error, a JSON-object with the success-field set to false and a message to explain the problem is returned.

```
{  
  "success":false,  
  "message":"Username busy"  
}
```

5.14.2 THE COMPONENTS

register.php

This file handles registration of new users. It checks that your chosen username is not taken, it encrypts your password, and returns a session you can use for later queries.

login.php

Expecting a username and password, login.php verifies your login details and returns a session for later queries.

session.php

Verifies a supplied session. It is used so the user doesn't have to type in a username/password again if he is already logged in, and to check that only logged in users can start games.

ip.php

This file is used to get the external (Internet) IP of the client. This is not strictly necessary since we only handle clients on the same WLAN, but it is also a simple way to check that the client is online and able to contact the game server.

checkgame.php

This file is called from 5-20 times while the client is looking for a game. The first time it is run, the client's session, IP and port is entered into the server's "looking for game"-table. On subsequent lookups it either returns nothing (just a status that you have not been paired) or the data you need to connect to an opponent (targetIP and targetPort).

Connectlooper.php

Is not supposed to be accessed by any client. This script is to be run on an interval basis, i.e. using crontab on GNU/Linux. Its purpose is to check the database for players that are

looking for a game, and connect these to. The connection occurs when the players data is moved from the “looking-for-game”-table, to the “game-running”-table. When the player connects after the server has paired it with another client, the IP and port of the other client will be returned.

5.14.3 THE DATABASE TABLES

Since the game server and supporting systems was not the primary focus of our project, the setup is quite simple. We have tables for users, users looking for a game, users currently in a game, and games that have been completed with information about their winners.

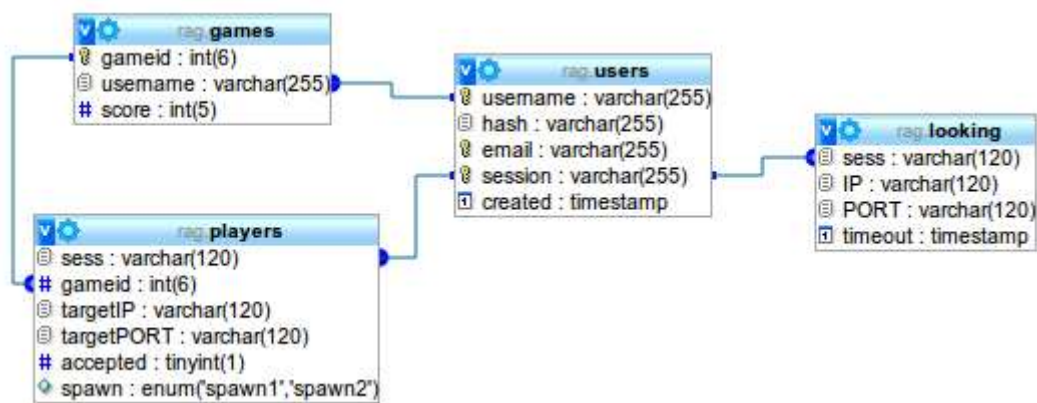


Figure 28: ER diagram, showing the tables in the database.

5.15 3RDPARTY

5.15.1 OPENGL

OpenGL is a API for displaying 2d and 3d graphics on a computer. Originally created by SGI and currently maintained by Khronos Group , a consortium consisting of members from IT- and Media- industry leading companies. (9)

OpenGL is a statebased system and the users enables the API to perform specific functionality by enabling states inside OpenGLs statemachine. As data progresses through this, the state dictates what to do. OpenGL was created for C and is not objective-oriented, even though it has become more and more Objective Oriented in the later years.

The Android version we use for our game, supports OpenGL ES 2.0, but we chose to use OpenGL ES 1.0, because the SDK only supports 1.0 not. For using OpenGL ES 2.0 you need to use the NDK.

We use OpenGL for communicating graphics to the screen. This means both storing data on memory available to the graphics chip on the phone and communicating to the same chip how it's going to look.

5.15.2 COLLADA

Collada is a fileformat created by Khronos Group. Its purpose is to store 3d model data in XML.

Collada defines an open standard XML schema for exchanging digital assets among various graphics software applications that might otherwise store their assets in incompatible file formats. (10)

5.15.3 ANDROID

Android is an open-source software-stack for mobile devices. It includes an operating system, middleware and several default applications. Android OS is based on the Linux Kernel. It runs on the Dalvik virtual machine featuring JIT compilation. (11)

When creating applications for Android, you usually work in Java, from the Android Software Development Kit, but you can also work in the Native Development Kit (Which is in C/C++).

We chose to work in the SDK and not the NDK for simplicity.

6 TESTING

We didn't decide initially on a specific way to test. We agreed that any system should be tested and approved by the programmer before implemented to the system.

6.1 TEST STRATEGY

Our method of testing our code was straightforward; after we had written a part of code to a workable state, we tested it quickly to see if it performed as expected. We tried to find common errors like null pointers, data not behaving like expected. When the code worked as we wanted it to, we moved on. If the code belonged to a bigger system, we would do these quick on-the-spot tests on all smaller systems and then do a more thorough testing when all the parts were completed.

By testing the system in parts, we prevented the final testing taking too long time, as all the smaller parts had already been tested before put together.

We also tested larger systems in a separate project, creating a setting for the test outside of the actual game. This helped us a lot to isolate the system and create specific situations that could provoke errors.

6.2 TESTING TOOLS/METHODS

We used two main methods for fixing errors in our code. First method is to use the Log class in `Android.Util.Log`. This class creates log data that can be read while running the game. This is fast to set up and gives you feedback from the code as the project is running, without having to stop the program.

The second method is to create breakpoints in the code and when the code stops at the breakpoint, go through the code step by step. This gives you a lot of details about what the code is doing *at this specific place in the code*, but doesn't give you information about what has happened before this spot.

6.3 GAME SERVER

The game server was tested using static HTML-forms that transmitted the same data as a normal client would. This way we could detect errors w/o having to run the request through an Android client.

7 CLOSING

7.1 DISCUSSION OF THE RESULTS

7.1.1 THE RESULT

As mentioned in the introduction, our main goal was to make a game that could be played. We wanted it to be technically functional, and entertaining. We believe we have reached this goal. The final version of the game can be played over network, and we all agreed that it is entertaining as well.

In order for two players to play the game together, they both need to be connected to the same wireless network, with an internet connection. The phones connect to the online server. Once the server has connected the phones, they use the wireless local network to communicate.

The user can quickly register a player account, and log in. When two players have registered an account, logged in, and clicked new game, the server quickly connects them, and a new game is started on both phones. If the connection is good, the response is quite good, and we almost don't notice that we play over network.

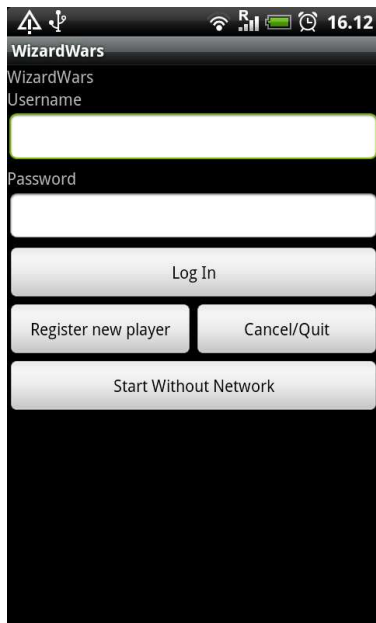
We added three different spells to the game. These have each their gesture attached to it. The spells were thunder spell, fire spell and ice spell. Fire spell is not working as it should, instead it's making the game hang. Ice spell doesn't do any damage for now, but thunder spell works as it should. It affects the health of the opponent, and decreases the player's mana. There is no way of "getting skilled" at casting a spell, it will always hit the opponent. The winner of a game is the one doing the highest number of spell casts.

When a spell is casted, the visual sign of this is an animation of the player, and a message appearing on the player's screen, telling which spell was casted. The health and mana will change as well. The opponent won't get a message about this.

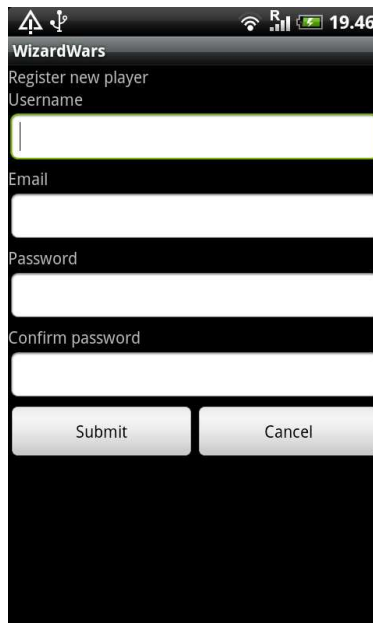
We implemented an offline game mode, which is basically the game without network support. This is actually a nice little feature, where the user can practice his skills, and try out the game.

To leave a game, the user simply presses the 'back'- or 'home' button. If another part of the Android system interrupts the game (e.g. the phone rings), the active game ends as well.

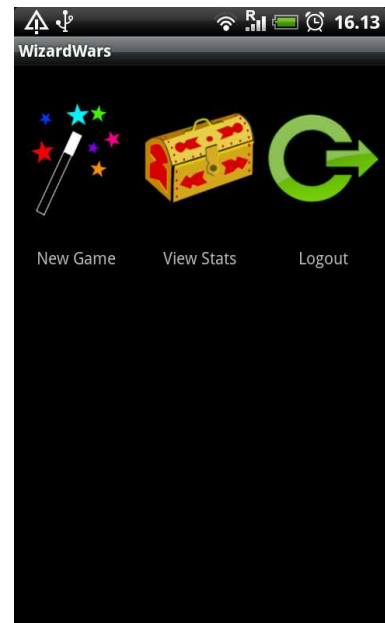
The login screen, registration screen, main menu, and the game screen are shown below.



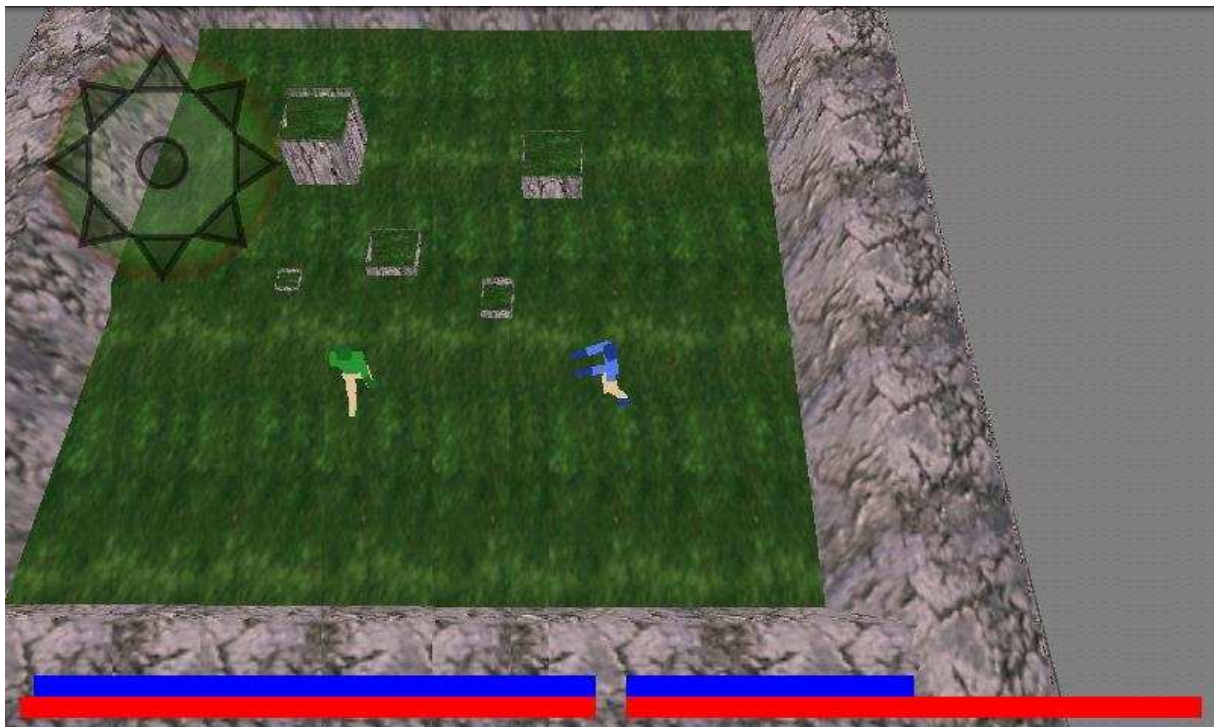
Log in screen



Registration



Main menu



An active game

7.1.2 DISCUSSION

We realized that since we first were going to make a game engine, and then implement the game play logic, this would take up a lot of time.

As time went by, we saw that because of our limited experience with game programming, we needed more time on each part of the project than we expected. Especially the development of the game engine became time consuming. We had a limited time table, so we had to cut back on some of the features described in the requirement specification. The differences between the requirement specification and what we accomplished are all due to the time limit of the project. These differences are described below. We tried to make the best out of the time we had, and not let this affect the most important parts of the project.

We decided to drop the feature where players can choose their opponent. Instead, the server finds two players looking for a new game, connects them, and the game starts on both phones. The two phones need to be connected to the same local network to play against each other, and to the internet to get connection with the server. This means that if more than two players are looking for a game, the opponent is random. Although we wanted the opportunity to choose opponent, and play over the internet, we think this solution works great as well.

The spell system is working as it should for the thunder spell. This shows that the system itself is working. However, we did not get the other two spells working properly. We believe this is a minor issue, since the process of adding new spells is relatively easy.

We did not prioritize the help section and the player statistics overview, so these features were not implemented. This does not affect the game itself, but they would have been nice features that would help improve the overall experience of the game.

We are pleased with how movement of a player works. The navigation icon is very responsive, and the collision system works pretty well. However, something is a little wrong with the bounding boxes for the walls, since the players collide with the walls before actually

May 27, 2011

touching them. We believe this is due to some differences in object reading and rendering. The collision handling between the two players and between a player and a box works fine.

7.2 WORK METHODS

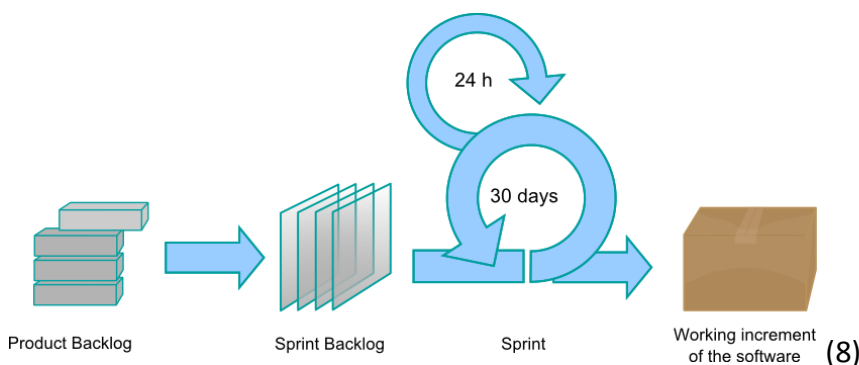
7.2.1 SCRUM

WHAT IS SCRUM?

Scrum is an agile development model. This means that everything is not decided prior to the development process, like in sequential models like waterfall, but through certain set of rules and methods, ideas can be changed or created along the way of the process. (12)

Before the development process starts, a product backlog is made. This contains a list of all the different parts needed to finish the product. Scrum uses sprints to split up the development process. These are usually from one week to four weeks long. At the start of each sprint you decide what parts you should work on for this sprint, by selecting tasks from the product backlog. This list of things to do is called a sprint backlog. After each sprint, a working preliminary copy of the product should be ready.

Working like this also opens for adding new tasks to the backlog as you progress or put non-finished tasks back into the backlog if deadline was not met. Every day, the team has a meeting to discuss what has been done since last meeting, and what should be done until the next meeting.



WHY DID WE CHOOSE SCRUM?

Since we had minimal experience with game programming, we would probably need to change both our design, ideas and time limits through the development process. This meant that a development model like waterfall would be a bad choice for us. We quickly agreed that we wanted to use an agile method as our development model.

When looking at the different methods we narrowed it down to 3 candidate methods; RUP, Scrum and Extreme Programming (XP).

RUP is a very advanced method, with a lot of tools and subsystems that can be used to work for smaller teams. But, all in all, RUP is best used in large teams and big projects.

XP is the opposite, it favors creation over documentation and also requires a very active project owner. XP could fit, if working extra on documentation, but project owner was not available at all times. Conclusion was that even though XP could be used, it would not be proper XP, so we chose to look elsewhere.

Scrum is a lot easier than RUP to both use and understand, as well as having better documentation methods than what XP has. We also believe that reporting to the project owner after each sprint is better suited for us and the owner, than having the owner physically represented in the development team at all times as XP requires.

Therefore, Scrum was our preferred choice of development model.

HOW DID IT WORK?

This was the first time we had used Scrum as a development model, so a little adaptation period was necessary. Initially, the sprints were supposed to last for two weeks. We realized that because of our limited experience with game programming and Android, we didn't

know exactly how long time we needed on each part of the game. Therefore, the sprints were shortened to one week, making it easier to plan what to do in the next sprint.

We made a product backlog at the start of the development process, containing all of the different parts we felt was needed for our game. At the start of each sprint, we had a short sprint planning meeting, deciding which parts we should do next. This was put in the sprints backlog.

As mentioned in the project plan, Appendix 7, our initial plan was to have both a sprint review meeting and a sprint retrospective meeting. In practice, these eventually got merged into one meeting, discussing the past sprint. We tried to document these meetings in the start, but eventually agreed that maybe this was unnecessary. Every morning we had a short daily scrum meeting, discussing what we had done since last time, and what to do this day. In the beginning of the project, we documented these meetings. Eventually, we realized that this was redundant, since these logs contained the same as our work logs.

We had planned to have a quick meeting once a week to review each other's code. This was not done that systematically, but rather done when needed. Finally, we had also planned to have a meeting with our supervisor at the start of every sprint. Since our supervisor had limited experience with game programming, we didn't feel the need for these meetings every week. Instead, we contacted him when it was needed.

We think that Scrum as the development model was an excellent choice for our project. Its rules and guidelines are easy to understand, and they ensure good documentation and communication within the group. Scrum is suitable even for a small group like this.

7.3 AFTERTHOUGHT

7.3.1 DO IT YOURSELF OR USE MIDDLEWARE?

Middleware, and third-party SDKs, are premade software packages tailored for use in other software packages.

When you develop your own software, you also learn a lot about how these systems work, something you do not learn to the same extent when using middleware. This is something that is important for us. We want to not only develop a game, but also have a maximum learning benefit from the project.

We decided that the learning benefit from creating our own system far superseded the benefit from using premade code. We didn't need an advanced rendering system or complex scene graph functionality, therefore many of the features in the middleware wouldn't be used.

7.3.2 MERGING SYSTEMS

When fitting together the different code parts, the network code and the game code had problem speaking properly together. This was due to different understandings about how these systems were going to work together. Better planning could have helped us to create code that fitted better together.

7.3.3 THE OpenGL/ANDROID HEADACHE

It seems that even though a phone is upgraded to a certain version of Android; It does not mean that it supports everything. We encountered this when trying to get our game running on a older phone: The graphics was corrupted! Testing it on our own phones(HTC Desire) and there was no problem. Eventually we did some tests and found out that the older phones only supported OpenGL 1.1. Now, OpenGL 1.1 was released in 1997 and OpenGL 1.2 was released in 1998. We are a bit surprised that a phone developed after 2009 does not support a API created in 1998, but the same one created in 1997.

7.3.4 ACTION SYSTEM

The action system was our attempt at creating a manageable, versatile and modular system for controlling the main interaction between players in our game: The spells. When starting on this part of the code, we decided that we wanted to have something that could handle anything, or at least not limit it by setting too strict rules for ourselves. We also stressed that we wanted this system to be highly modular, meaning that we wanted it to easily be added upon. This proved to be a two edged sword: On one hand you have a system that can be built upon for later projects, or create new interaction methods for the this project if time allows it, on the other hand you have a system that is too generic and hard to understand. In other words: The system could have been made a lot smaller and would still perform just as good.

A system that handles player interaction will always have a higher level of complexity than many other systems, just because it is interacting with the player. As a human you expect something to behave in a certain way and you expect to get feedback when something happens. This is part of the reason why these systems gets so complex.

The underlying idea of the system isn't bad, but as implementation continued and especially, when the implementation of the action system met the other systems (Especially the

Network system), the system started to become more and more rigid, moving away from the idea that it was going to be able to handle anything.

What we really learned from creating this system was how important it is to plan out such systems before implementing. Once you develop a system where the player is interacting with, or through, the computer you need to set clear boundaries to that system. Never to make a system that “can handle anything”.

We didn’t really design this system before implementing it, and that was a big mistake. It would have been a lot simpler to design this on paper before actually writing the code for it. The system as a whole is a Finite State Machine (FSM) and is easily drawn up on paper. We realized this halfway through the implementation of the system and from there on treated the system as a FSM, which made the implementation a lot easier.

7.3.5 COLLADA VS .OBJ

When beginning the coding of the scene graph and the parser tool, we were converting from 3dsmax to a file type called OBJ (13). Once we thought the systems were implemented, we had serious problems with displaying the 3d models properly in our game. After going back and forth, trying to figure out what was wrong, we eventually found the error.

Obj files are saved by using separate indexes for vertices, normals and texture coordinates. These indexes are optimized separately without regard to how the others are structured.

OpenGL requires the data to be sorted so that you can use only one index list, meaning all data be sorted so that the same index in all 3 lists, deal with the same point in space.

We now had 2 choices. Read in the OBJ file and then restructure the data as needed, or switch to a new file format all together. Restructuring the data *could* introduce more bugs and more trouble, but we wouldn’t have to change much of the existing code. Switching file

format would mean rewriting the existing code, but would not require extra steps when reading data.

The new file format we found to suit us best was the file format Collada (10). This file format stored data in a way that allowed us to transfer the data directly into arrays without any restructuring.

We chose the latter. Restructuring the data *could* be faster, but the end result was not certain. Choosing a new file format that we *knew* would work and rewriting parts of the parser(Parts we had already implemented for .obj files) gave us a fairly good overview of the amount of work needed. Looking back at this point in time, we feel we made a very good decision here, and the time used for implementing the new file format was minimal.

7.4 CONCLUSION

We started this project with great enthusiasm, all agreed upon following scrum to the letter, by doing all meetings, planning as it should be and follow up code with proper testing regimes. After progressing through the project for a while, we started to see that not every project needs all that scrum defines, and testing doesn't need to be done in a set way.

When working on a bachelor project it is easy to think early, that you have lots of time. Well, you don't. A computer game, of any size, is a big project. It consists of several systems, spread across many layers of code and communication between them needs to be fast, manageable and safe.

We knew early on, that this project was going to be a lot of work, all up to the end. Therefore we started by documenting everything and following scrum to the letter. Daily meetings, two week sprints, planning meeting, review meeting. Everything.

About 1/3rd into the project we also started noticing that daily scrum meetings consisted usually of the same logs. We felt that saying the same thing every day was getting pointless. So we changed from daily scrums to every 2 to 3 days logs and weekly code discussions. From a development perspective this worked great, but we do see that for documentation purposes, this wasn't the best idea.

We did however, have an extra layer for documentation. All members of the group agreed to keep a work log that was to be updated frequently. This was done with varying success, but in general most parts of the project has been logged.

2/3rds into the project, one of the project members left, to be gone until the very last days of the project. This took a huge strain on the developing process. Not so much on the actual coding, but the documentation went from a structured method, to a vocal or whiteboard method. We did take pictures of pretty much all whiteboard meetings, but we do feel that we should have continued to use the scrum method.

One thing that we felt missing from scrum, that we tried to implement, was code meetings. Scrum does have Sprint review meetings, where you can talk about how the last Sprint was.

But it doesn't necessarily mean that this meeting should be spent on reviewing code, it should be used to talk about the process *and* code, not the code alone.

Looking back, we see that our work method wasn't a pure scrum method. But the changes we did were done intentionally. When we saw that we were doing something that wasn't for the sake of the project, but for the sake of the process; we discussed it together and found a solution where everyone agreed that this would be best.

We learned a lot from this project. Writing and designing a project of this size, forces all to work together and write code that's usable by others. Creating a game also requires a lot of algorithm and system engineering techniques that we learned in theory from previous courses in our degree. Using what we learned before, in an actual software project, gave us better insight and perspective of why we are here. All of us are looking forward to starting our careers with new found knowledge.

WORKS CITED

1. http://www.amobil.no/artikler/dette_er_favorittmobilene_til_netcoms_kunder/80762 .
amobil.no. [Online]
2. http://www.amobil.no/artikler/her_er_de_25_mest_populaere_mobilene/80621/2.
amobil.no. [Online]
3. http://en.wikipedia.org/wiki/Back-face_culling. *wikipedia*. [Online]
4. [book auth.] Mike McShaffry. *Game Coding Complete, 3rd edition*. s.l. : Delmar, 2009.
5. **Khronos**. http://www.opengl.org/wiki/Vertex_Buffer_Object. *opengl.org*. [Online]
6. **Wright Jr, Richard S., et al., et al.** *OpenGL SuperBible: Comprehensive Tutorial and Reference*. s.l. : Addison Wesley, 2010.
7. <http://download.oracle.com/javase/6/docs/api/java/io/Serializable.html>.
www.oracle.com. [Online] oracle.
8. http://no.wikipedia.org/wiki/Fil:Scrum_process.svg. *wikipedia.org*. [Online]
9. <http://en.wikipedia.org/wiki/OpenGL>. *wikipedia.org*. [Online]
10. <http://en.wikipedia.org/wiki/COLLADA>. *wikipedia.org*. [Online]
11. <http://developer.android.com/>. *www.android.com*. [Online] Google.
12. <http://scrummethodology.com/>. *scrummethodology*. [Online]
13. http://en.wikipedia.org/wiki/Wavefront_.obj_file. *wikipedia*. [Online]
14. **Dalmau, Daniel Sánchez-Crespo**. *Core Techniques and Algorithms in Game Programming*. s.l. : New Riders, 2004.

8 Appendices

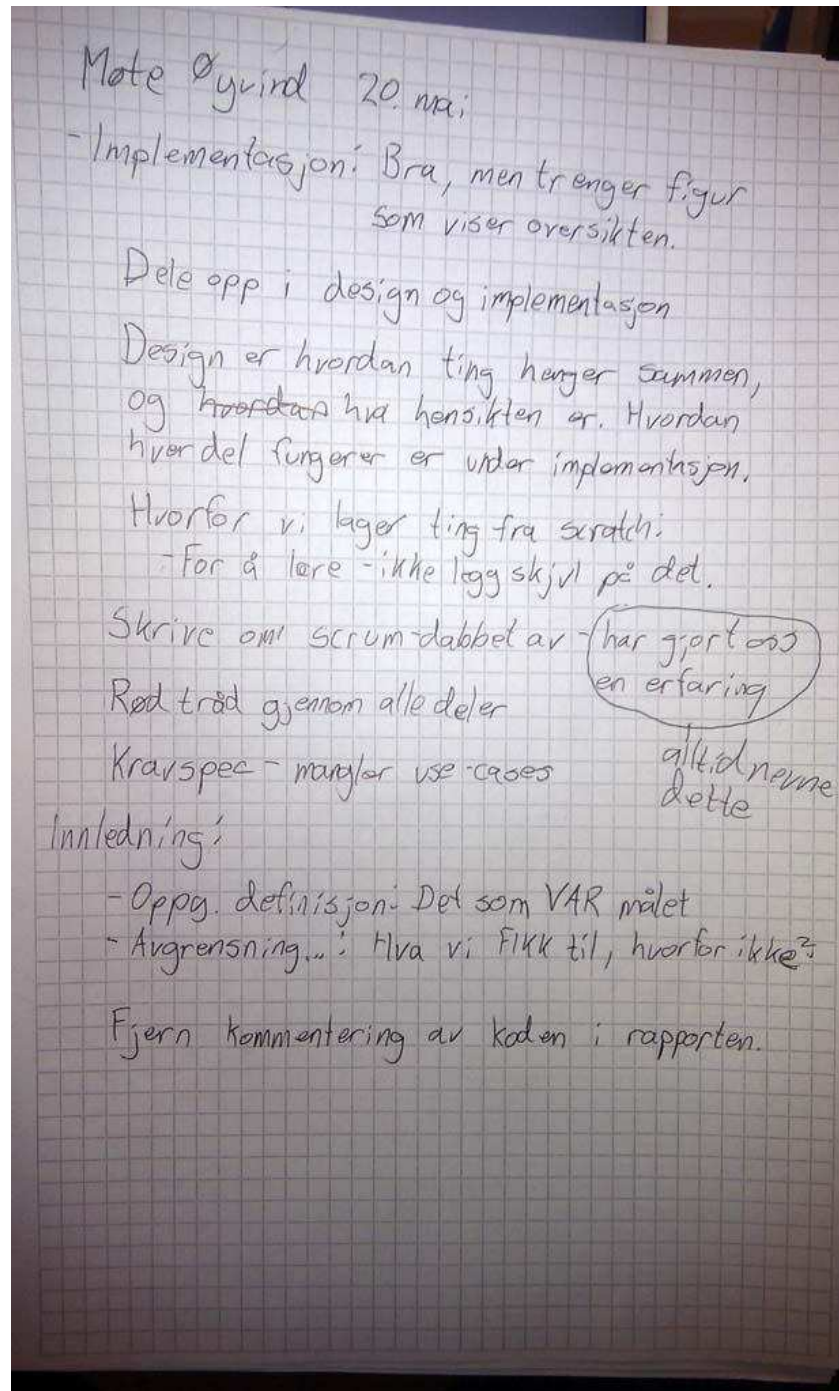
CONTENTS

Appendix 1: Project diary.....	115
Meeting 20.05.2011 - Feedback Øyvind.....	115
Appendix 2: Worklogs.....	135
Work log for Ole Marius Kohmann.....	146
Worklog Anders Einar Hilden	152
Appendix 3: Status reports	155
Appendix 4: Scrum meetings	157
Sprint 7 (29.03.2011 - 04.04.2011).....	157
Appendix 5: Daily scrums.....	161
Appendix 6: First Technical description What we need	168
Appendix 7: Project Plan (Pre project report)	170
5 Table of contents□	170
6 1. GOALS AND BOUNDARIES	171
1.1 Background	171
1.2. Goals.....	171
1.3. Boundaries	171
7 2. SCOPE.....	173
2.1. Project description	173
2.2. Scope.....	173
8 3. Project organization.....	174
3.1 Employer and supervisor	174

3.2. Responsibilities and roles.....	174
3.3. Group rules and routines	174
3.3.1 Group rules	174
3.3.2 Routines	174
3.4 Resources	175
9 4. Planning, meetings and reporting.....	176
4.1. System development model	176
4.2. Meetings	177
4.3 Status reports.....	177
10 5. Quality assurance	177
5.1 Testing.....	177
5.2 Code meeting.....	178
Once a week we will have a group meeting where we read through each others code. This way we assure that the quality of the code is maintained. 6. Gantt chart Since we are using Scrum as our development model, the development period of the project will be divided into sprints. We set one sprint to be two weeks. This gives us four sprints total. At the end of each sprint we will have a working product.	178
Appendix 8	180

Appendix 1: Project diary

Meeting 20.05.2011 - Feedback Øyvind



MEETING MONDAY 10.01.2011

Attendees:

Ole Marius Kohmann, Jon Lande, Simon McCallum, Jayson Mackie.

Topics:

Supervisor.

Project scope.

Project technical & creative brainstorming.

Project goals for the students.

Roles.

Details:

Concerns were raised about having a supervisor who is not experienced with game development and game programming. Simon advised Ole Marius to contact Rune Hjelsvold for further discussions about this issue.

The size of the project was discussed, there are some concerns related to Jon and Anders Einar not being experienced with game programming already, but the group was quickly assured by Simon and Jayson that this would not be such a big problem when they get hands on experience with this kind of coding.

Ole talked about his ideas for the project being a multiplayer game in 3d, where players can fight eachother as wizards. The game would be controlled by using the trackball and doing gestures on the screen itself. There is a wish for doing this in a realtime style gameplay, but we suspect it is more feasible to do this in a turn-based style game.

The group talked about what the students were aiming for in terms of grading and was explained what it is expected to put into the project for the different grades.

The students said they were going to aim for an A.

Being maybe a bit too early to set in stone the roles(As Anders Einar was not present), it was agreed by all that it would be best if Ole Marius took the role as lead designer.

Notes:

Anders Einar Hilden was not in the meeting, due to conflicting schedules. This was informed of before the meeting time was set.

MEETING WEDNESDAY 12.01.2011

Attendees:

Ole Marius Kohmann, Jon Lande, Simon McCallum, Jayson Mackie

Topics:

Supervisor part 2

Room

Details:

The students have talked together around the supervisor issue and decided that it would infact be best if we did have the originally assigned supervisor Øyvind Kolloen and not try to find someone who are more experienced in the game programming field.

Having Øyvind Kolloen as supervisor will give us a different perspective than that of a game programmer. We now see that this is of great benefit to us.

Ole Marius has sent an email to Rune Hjelsvold to explain our new view on this.

The room we have been assigned to is not a good place to create a project of this complexity. This project needs a work area that allows the students to sit in deep concentration for several hours without being disturbed. Sitting in room A112 together with 4 other groups and a total of 12 people in the room is a great concern to the students.

Ole Marius sent an email to the school, explaining our concerns.

Notes:

Anders Einar informed today that he is coming back on monday 17th january.

MEETING WEDNESDAY 19.01.2011

Attendees:

Ole Marius Kohmann, Jon Lande, Anders Einar Hilden.

Topics:

Finalise roles.

Divide responsibility.

Specify idea.

Project contract.

Project plan.

“How do we want to work?”

Agree on coding style.

What tools do we use?

Agree on version control.

Scrum/RUP/XP?

.....?

Details:

Finalise roles:

Ole is lead designer.

Jon is project manager..

Anders Einar is technical supervisor.

Responsibilities:

Anders Einar is responsible for the website and the version control.

Jon is responsible for the project reports

Ole is responsible for the idea.

Project contract:

Ole will talk to Torunn Linneberg regarding a updated project contract that contains an agreement around the source code and its its right of use/ownership.

Project plan:

Jon will write up a suggestion to the main headlines for the project plan by tomorrow.

Group agrees that we need to talk to Øyvind tomorrow for more details around how the project plan should be structured and its content.

Scrum/RUP/XP?:

Scrum.

Anders Einar will research Scrum and present it to the rest of the group on friday 21.01.11.

“How do we want to work?”:

Scrum meeting: Mondays.

Midweek meeting: Thursdays.

Workhours: Tuesday - friday: 09.00- 15.00.

Codingstyle:

Standard Java capitalization.

Tabs, not spaces.

```
public class BookShop {  
  
    public void getBook(String title) {  
  
    }  
  
}
```

Tools

git

Eclipse Helios

EGit

Android SDK (Android 2.2)

Idea:

- Android game for Android 2.2.
- Multiplayer.
- 3d (openGL) top down, perspective view.
- Competition oriented.
- 1 vs 1, arenastyle gameplay.
- using gestures to control what skills to use.
- if realtime; use trackball to move around.

Next meeting:

Anders Einar will research possible game servers w. regards to complexity, possibilities, and gameplay (turnbased/realtime).

Ole will continue to port his code from game programming and research what is needed for it to go 3d.

Jon will make a blueprint for the project description delivery that is due 29th january.

Notes:

Øyvind Kolloen has been gone since friday. The group has been informed that he will be back tomorrow(Thursday 20th january).

MEETING FRIDAY 21.01.11 WITH SUPERVISOR

Attendees:

Ole Marius Kohmann, Jon Lande, Anders Einar Hilden, Øyvind Kolloen.

Topics:

Gameserver

.....?

Details:

OMK started the meeting by informing ØK that (and why) we had settled on him as a supervisor.

OMK: What is the target of the bachelor-thesis - the product or the project rapport?

ØK: There should be a guiding schema somewhere on the HiG-web, but that in the end, the overall delivery is judged.

OMK: What is most important regarding the code - smart code or dirty code that does it's job?

ØK: Sensible, reusable code is important. Use libraries if they fit (not too complex, not too useless), you will not be rewarded for reinventing the wheel.

OMK/AEH: Should we make our own UDP-java server, or find and use a framework already written ?

ØK: Overkill (mega framework) is stupid. Check what is out there - if you find something that matches your needs or is simple to redesign, use it. If you can't find anything with a fitting complexity, write something yourself - but whatever we choose, list your reasons for doing either in your rapport.

OMK: What should we "use" ØK for?

ØK: Project plan, progress plan, work methods. Game questions should be directed at Jayson/Simon. You should consider contacting Kjell Arne Refsvik (mobile development) to see if he can be a resource as well. He might be able to help you with basic GUI designs for small screens etc.

OMK/JL: Can we deliver an updated project plan later after we have adjusted/found problems with it, or is it "locked" ?

ØK: It is OK to update it later, after you have found some traps or used somewhat longer on a thing than planned - but sleeping all of January/February will not be tolerated. Remember to write the rapport as you work - it is stupid to write it all the last two weeks.

OMK: Mixing pair programming (XP) with scrum iterations?

ØK: Indeed possible. Don't mix it all. It should be possible to follow your plan. Don't pick all the "good" bits or drop all the hard ones. Easy projects (netshops, timetables etc) are too simple. SQL queries with PHP on top) a normal student will get a C. Technically heavy projects

can get a B with the same amount of work.

Drop “obvious” things in in you work/rapport, remember to include the interesting bits.

Explain your targets and limits. (i.e android 2.2, 1ghz cpu, x ram).

Time-use: They have enough experience with bachelor-thesis’ that they know if a low volume of hours i a genius working fast, or a average working to little.

Project plan: Should have a scrum product queue, and a somewhat accurate plan of attack. Suggestions for dates, but these will change.

Meetings v. ØK: Every 14. days, preferably after a SCRUM-sprint is completed. If we submit progress papers to ØK at the morning, a meeting after lunch is enough time for him to read them.

ØK shedule: Mon-Wed: Free all day. Tuesday: Busy after lunch. Friday: Busy before lunch.

Class-tree, use-cases, gaunt (accurate) in specification. Create a rough schedule of you project, what you want to have finished after each sprint, etc.

Remember to test as-you-code. Design the test first, then write the code. If the test fails, the code is not complete.

MEETING MONDAY 31.01.11

ATTENDEES:

OLE MARIUS KOHMANN, ANDERS EINAR HILDEN, JON SÆTHEREN LANDE

TOPICS:

Coming scrum

Sprint length

Sprint work methods

Notes:

1. Sprint length: 2 weeks, might need to change to 1 week.

2. Sprint work methods.

1 day to research & proof of concept.

4 days to reach feature complete.

5 days to finish & test.

3. Coming scrum.

Tasks that will be worked on the coming sprint:

Server:

Highlights:

Rudimentary lobby

Message pump.

Game server thread.

Test Requirement:

Be able to echo messages.

Be able to add and remove users to lobby and remove timeout users.

Match two random players in lobby by request.

Start new game-thread. by tcp query.

start game after both users have joined game thread.

Relay/use packages.

stop game when player loses.

Update stats.

Return users stats on request(TCP).

Sane check packages in game thread.

ObjParser

Test Requirement:

1. be able to read "tags" in a .obj file and its coming data.
2. Be able to create nodes of correct type and store data correctly in them.
3. be able to read .mtl file in same way as .obj file.

SceneGraph

Classes:

SceneGraph

Node

TranslateNode

RotateNode

ModelNode

GLTraverser

ClippingTraverser

Test Requirement:

- Be able to create a SceneGraph correctly read from objparser file.
- Be able to traverse through the tree and display info about each nodes location and data.

Renderroutine

Player

Work Description:

Gui

Work Description:

Next meeting:

Notes:

MEETING WEDNESDAY 02.02.11

Attendees:

Ole Marius Kohmann, Jon Sætheren Lande, Anders Einar Hilden

Topics:

Anders Einar expressed concerns regarding network code & and problems with connecting two phones together through internet. Main issue is regarding peer to peer connections and how to connect through routers in each end. The group talked about several solutions and came to the conclusion that Anders Einar will talk to Jayson/Simon and also do more research before we make our final decision.

SKYPE MEETING 16.02.11 (SHORT)

Unfortunately, Ole Marius had to stay home with sick children both tuesday and wednesday this week, so the meetings at the end of this sprint and start of the next sprint had to be postponed.

Since Ole Marius had to stay home, we had a short meeting over Skype.

We decided we will try to have a meeting with both the supervisor and the employers this week, in addition to our scrum sprint meetings. We didn't finish all this sprints parts of the project, like we planned. Since most parts of the project is new to us, we need to do a lot of research. This also means that we have a little trouble deciding on how long time we need for each task. Therefore, we are thinking of changing the sprint length to one week, instead of two.

May 27, 2011

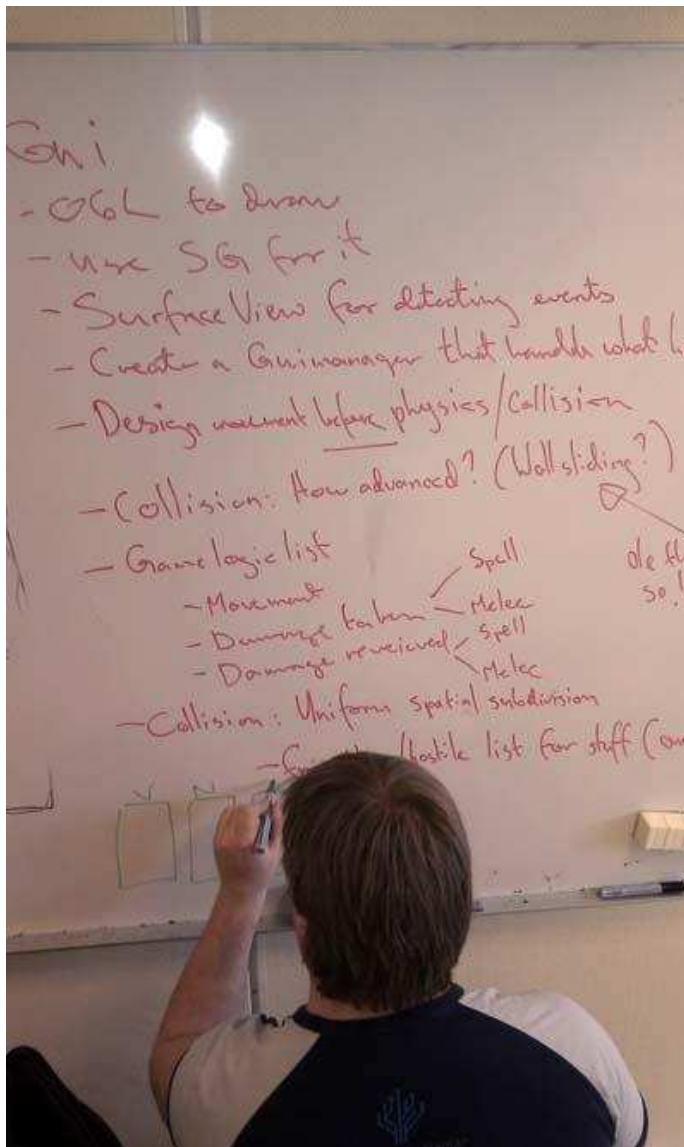
MEETING TUESDAY 01.03.11

Attendees:

Ole Marius Kohmann, Jon Sætheren Lande, Jayson Mackie

We had a meeting with Jayson concerning the progress of the project so far, and how to proceed. Anders Einar was sick, and had to stay home.

We demonstrated what we had done so far, and got some positive feedback. When looking at our work/progression schedule, Jayson suggested some changes in the order to do things. He also had valuable input on how to organize the game logic.



Notes:

PLANNING MEETING WEDNESDAY 02.03.11

Attendees:

Ole Marius Kohmann, Anders Einar Hilden, Jon Lande

Planning meeting where we decided what tasks we will focus on next.

Notes:

Pr. 02.03.2011:

TODO Next:

Ingame GUI / HUD (Heads up display)

- Player info
- Hp
- Mana
- ++ (?)
- Opponent info
- Event handling

Player controls

- Player movement
- Player actions (gestures)

Update routine (controller / controller list)

- Design method for update routine
- System for handling all movement & rotations of all objects in current level
- Controller system for the object

Movement design

- Need to decide how the player moves, what is allowed and what restrictions are involved.
- Both from a technical perspective and design perspective

Collision / physics

- Decide on collision features
- Scope of physics system:
 - Player/opponent ground collision
 - Player/opponent wall collision
 - Spell travelling physics

- Out of boundaries

08.02.2011

After a meeting with Jayson, we found out that the format of the .obj files we were using when reading 3D objects was useless for us, because of the way OpenGL interpreted the texture data. Therefore we decided to use the xml based Collada format (.dae files) instead. This meant some re-writing of the code for parsing game objects, and adding them to the scene graph.

PLANNING MEETING WEDNESDAY 15.03.11

Attendees:

Ole Marius Kohmann, Anders Einar Hilden, Jon Lande

Topics:

Sprint review

Sprint retrospective

Daily Scrum log - necessary?

Sprint Planning meeting

(See scrum meetings for scrum related topics)

Daily Scrum log:

Group agrees that daily scrum logs are less needed for the project. Team also agrees to put more effort into keeping our own work logs updated and feel that this is enough to track our

progress and time put into the project.

MEETING THURSDAY 17.03.11 WITH SUPERVISOR

Attendees:

Ole Marius Kohmann, Jon Lande, Anders Einar Hilden, Øyvind Kolloen.

Topics:

Status update

Discussing progression

Network

Presentation

Details:

OMK gave Øyvind a status update, telling about our progress. We were concerned that not all of our meeting are documented properly, but Ø said this was ok as long as we got the most important.

We decided that we will write three status reports to supervisor and employer in all. We have sent one, and the two next will be at the start of April and start of May.

Øyvind was a little concerned about the network/multi-player part of the game, and warned us that it usually takes longer time than we think. Since Anders Einar will be gone for a month towards the end of the project, and he is the network guy, we can't wait too long before focusing on network.

We were wondering if the fact that we don't have a detailed plan of the gameplay yet was a major issue. This was not a problem, since what we have done so far could have been used

for almost any game. However, we won't wait too long before focusing on this.

We discussed that the group members have different approaches to code debugging. Øyvind suggested that we mixed our approaches for most effective debugging.

Ole M was concerned if we should have used more third party libraries for things like the scene graph. This was okay to make ourselves, as long as we argued why in the report.

We discussed short the presentation, and Øyvind told us to make a idiot-proof presentation of the game, not try to play in real time.

All in all, Øyvind was pleased with our progress, and was not very concerned.

APPENDIX 2: WORKLOGS

WORKLOG FOR JON LANDE

Week 1:	12 hours
Research android	6 h
Research game programming	6 h
Week 2:	11 hours
Mon 10.01.2011:	6 h
Meeting with Simon/Jayson/Ole M	3 h
Meeting with Ole M	1 h
Research game programming	2 h
Wed 12.01.2011:	2 h
Tools installation / learning	2 h
Fri 14.01.2011:	3 h
Research android dev	3 h
Week 3:	24,5 hours
Mon 17.01.2011:	3 h
Getting room / computer for project	1 h
Research android dev	2 h
Tue 18.01.2011:	3 h
Setting up desktop computer and installing software	3 h

May 27, 2011

Wed 19.01.2011:	6 h
Troubleshooting computer	1 h
Group meeting / project planning	5 h
Started on the project plan	< 1 h
Thu 20.01.2011:	6 h
Working on project plan	6 h
Fri 21.01.2011:	6,5 h
Project plan	4 h
Research game programming	1 h
Meeting with supervisor	1,5 h
Week 4:	29,5 hours
Tue 25.01.2011:	6 h
Project plan	4 h
Research android/game prog	2 h
Wed 26.01.2011:	7,5 h
Project plan	5,5 h
Research/testing Android dev	2 h
Thu 27.01.2011:	9 h
Research Android dev	9 h
Fri 28.01.2011:	7 h
Project plan	4 h
Opengl research	3 h
Week 5:	28 hours
Monday 31.01.2011:	4 h

May 27, 2011

Working on project contract	2 h
Started writing obj parser	2 h
Tue 01.02.2011:	7 h
Writing obj parser	6 h
Learning openGL from OM	1 h
Wed 02.02.2011:	7 h
Working on obj parser	7 h
Thu 03.02.2011:	4 h
Writing .mtl parser	4 h
Fri 04.02.2011:	6 h
Reading about OpenGL	6 h
Week 6:	27,5 hours
Tue 08.02.2011:	8 h
Researching sounds in Android	8 h
Wed 09.02.2011:	6,5 h
Started on GUI	6 h
Database planning	0,5 h
Thu 10.02.2011:	6,5 h
Worked on GUI / login	6,5 h
Fri 11.02.2011:	6,5
GUI (login / register)	6,5h
Week 7:	23 hours

May 27, 2011

Tue 15.02.2011	8 h
Working on login and register	8h
Wed 16.02.2011	
Register new player	7 h
Thu 17.02.2011	2 h
Register	2h
Fri 18.02.2011	6 h
Playing with OpenGL in Android	6 h
Week 8:	25 h hours
Tue 22.02.2011	6 h
Scrum meetings, sprint planning	4 h
Scene graph planning	2 h
Wed 23.02.2011	6 h
Started on building the initial scene graph from .obj files	6 h
Thu 24.02.2011	6 h
Creating geometry nodes from .obj files, writing them to binary file (on computer), then reading nodes from this file (on phone)	6 h
Fri 25.02.2011	7 h
Continued on sg I/O, debugging parser	7 h
Week 9:	30,5 hours
Mon 28.02.2011	3 h

May 27, 2011

Scrum meetings 1 h

Debugging .obj file parsing 2 h

Tue 01.03.2011 8 h

Fixed the .obj parsing and rendering, 8 h
so game objects are shown correctly on
phone

Wed 02.03.2011 7 h

Work planning meeting 1 h

Adding geometry node to scene graph, 6 h
adding a texture node, then writing
the whole sg to file (on computer), reading
back in (on phone)

Thu 03.03.2011 6,5 h

Traversing sg for initialization, trying to get 4,5 h
texture mapping to objects working correctly

Reading about collision detection 2 h

Fri 04.03.2011 6 h

Reading collision detection 6 h

Week 10: 20 hours

Mon 07.03.2011

?

Tue 08.03.2011 6 h

Project planning, 6 h
debugging texture mapping

May 27, 2011

Wed 09.03.2011	4 h
Collada file parser	3 h
Lecture: Writing bachelor report	1 h
Thu 10.03.2011	6 h
Got collada file parser working	6 h
Fri 11.03.2011	4 h
Reading	4 h
Week 11:	17 h
Tue 15.03.2011	6 h
Scrum meetings	2 h
Reading and planning code	2 h
Planning collision detection	2 h
Thu 17.03.2011	8 h
Meeting with supervisor	1 h
Collision detection	7 h
Fri 18.03.2011	3 h
Collision detection	3 h
Week 12:	30 h
Tue 22.03.2011	8 h
Collision detection	8 h
Wed 23.03.2011	4 h
Collision detection	4 h
Fri 25.03.2011	6 h

May 27, 2011

Collision	6 h
Sat 26.03.2011	4 h
Collision	4 h
Sun 27.03.2011	8 h
Collision	8 h
Week 13:	37 h
Mon 28.03.2011	3 h
Collision	3 h
Tue 29.03.2011	10,5 h
Collision handling	10,5 h
Wed 30.03.2011	3 h
Collision	3 h
Thu 31.03.2011	7,5 h
Collision	7,5 h
Fri 01.04.2011	
Collision	8 h
Sat 02.04.2011	2 h
Collision handling debugging	2 h
Sun 03.04.2011	3 h
Collision handling debugging	3 h
Week 14:	20 h

May 27, 2011

Mon 04.04.2011	3 h
Collision	3 h
Tue 05.04.2011	7 h
Modifying collada parser	7 h
Fri 08.04.2011	9 h
Working on player movement	9 h
Week 15:	24 h
Tue 12.04.2011	7 h
Planning meeting	1 h
Collision	6 h
Wed 13.04.2011	7 h
Collision debugging	7 h
Thu 14.04.2011	10 h
Collision debugging	10 h
Week 16 - Easter:	
-	
Week 17:	26 h
Mon 25.04.2011	7 h
Working on animation system for player	7 h
Wed 27.04.2011	7 h
Network movement, animation	7 h
Thu 28.04.2011	4 h

May 27, 2011

Network movement	4 h
Fri 29.04.2011	8 h
Network movement, collision, scenegraph	8 h
Week 18:	39 h
Mon 02.04.2011	8 h
Debugging scene graph/collision	8 h
Tue 03.04.2011	
Debugging scene graph/collision	6 h
Wed 04.04.2011	7 h
Starting on the report	6 h
Writing status report	1 h
Thu 05.05.2011	7 h
Report: Collision system	7 h
Fri 06.05.2011	7 h
Report: Loading of Scene graph	4 h
Code: Working on animation system	3 h
Sat 07.05.2011	4 h
Report research	4 h
Week 19:	25 h
Tue 10.05.2011	7 h
Report: Code organization and data structures	7 h
Wed 11.05.2011	5 h

Debugging wrong displaying of objects in emulator and other units	5 h
Thu 12.05.2011	7 h
Got walking animation working	6 h
Report: Work methods	1 h
Fri 13.05.2011	6 h
Planning report, Report: Collision	6 h
Week 20:	49 h
Mon 16.05.2011	5 h
Report: Work methods, collision	5 h
Wed 18.05.2011	8 h
Report: Humans, serialization, initialization	8 h
Thu 19.05.2011	8 h
Report: Introduction	8 h
Fri 20.05.2011	8 h
Meeting with supervisor	1 h
Report: Dividing up code desc. & design, introduction ++	7 h
Sat 21.05.2011	8 h
Modifying collada parser (translating objects correctly) + sg reading	8 h
Sun 22.05.2011	12 h
Finalizing the coding, game result	12 h

conroller, reading spawnpoints

Week 21	44 h
---------	------

Mon 23.05.2011	9 h
----------------	-----

Code: Testing network code	2 h
----------------------------	-----

Report: Requirement specification	7 h
-----------------------------------	-----

Tue 24.05.2011

Report: Requirement specification	8 h
-----------------------------------	-----

Wed 25.05.2011	12 h
----------------	------

Report: Requirements & introduction	12 h
-------------------------------------	------

Code: Fixed enemy animation

Thu 26.05.2011

Report: Finalizing	15 h
--------------------	------

WORK LOG FOR OLE MARIUS KOHMANN

Week 1:	27 hours
Research Java	12 hours
Research android	10 hours
Research game programming	5 hours
Week 2:	32 hours
monday 10.01.2011:	8 hours
Meeting with Simon/Jayson/Jon	3 hours
Meeting with Jon	1 hour
Research android	4 hours
Tuesday 11.01.2011:	7 hours
Research android	4 hours
Research SVN/GIT/CVS	3 hours
Wednesday 12.01.2011:	8 hours
Research game programming android	4 hours
Meeting with Simon/Jayson	2 hours
internet meetings	2 hours
Thursday 13.01.2011:	6 hours
Tools research	3 hours
Tools installation	1 hour
Research android	2 hours
Friday 14.01.2011:	0 hours
No work	
Saturday 15.01.2011:	0 hours
No work	
Sunday 16.01.2011:	3 hours
Code converting(c++ to java)	3 hours
Week 3	23.5 hours
Monday 18.01.2011:	5 hours
Administrative tasks(room, computers++)	1 hours
Android programming research	2 hours
Code converting(c++ to java)	2 hours
Tuesday 18.01.2011:	0 hours
No work	

May 27, 2011

Wednesday 19.01.2011:	6 hours
Group meeting / project planning	5 hours
Working on the projectplan	1 hours

Thursday 20.01.2011:	6 hours
Working on project plan	3 hours
OpenGL for Android research	2 hours
Code converting(c++ to java)	1 hours

Friday 21.01.2011:	6,5 hours
Project plan	4 hours
Research Android	1 hours
Meeting with supervisor	1,5 hours

Week 4 **31.5 hours**

Monday 24.01.2011: (Normal school)	0 hours
---------------------------------------	---------

Tuesday 25.01.2011:	8 hours
Meeting with Jayson(Scenegraph)	1 hour
Working on the projectplan	5 hours
Code research	2 hours

Wednesday 26.01.2011:	7.5 hours
Project plan	4 hours
Meeting Jayson/Simon SG/contract	1 hour
Research/testing Android dev	2.5 hours

Thursday 27.01.2011:	8 hours
Projectplan	3 hours
SceneGraph research	3 hours
Lecturing others on OGL/SG	2 hours

Friday 28.01.2011:	8 hours
Projectplan	5 hours
SceneGraph code planning	1 hours
Scrum backlog planning	2 hours

Week 5 **31 hours**

Monday 31.01.2011:	4 hours
Project contract work	2 hours
Scrum backlog planning	2 hours

Tuesday 01.02.2011:	6 hours
---------------------	---------

May 27, 2011

SceneGraph coding/Researching	3 hours
Scrum meeting	1 hour
Teaching JSL some OGL	1 hour
Java research regarding SceneGraph	1 hour

Wednesday 02.02.2011:	7 hours
Rendering coding(SG Traversing)	2 hours
SceneGraph coding	4 hours
Various meetings	1 hour

Thursday 03.02.2011:	6 hours
SceneGraph research	2 hours
SceneGraph coding	2 hours
Rendering coding(SG Traversing)	2 hours

Friday 04.02.2011:	0 hours
School work	

Saturday 05.02.2011:	5 hours
SceneGraph research	2 hours
SceneGraph coding	3 hours

Sunday 06.02.2011:	5 hours
SceneGraph research	3 hours
Rendering coding(SG Traversing)	2 hours

Week 6	18 hours
---------------	-----------------

(Worked from home this week)

Monday 07.02.2011:	3 hours
SG Traversing code/research	3 hours

Tuesday 08.02.2011:	5 hours
SceneGraph Traversing code	5 hours

Wednesday 09.02.2011:	3 hours
SceneGraph	

Thursday 10.02.2011:	3 hours
SceneGraph	

Friday 11.02.2011:	2 hours
SceneGraph	

Saturday 12.02.2011:	1 hours
SceneGraph	

Sunday 13.02.2011: 1 hours

Week 7(Winter vacation) 1 hours

Monday 15.02.2011: 1 hours
Meeting
(Schoolday)

Tuesday 16.02.2011: 0 hours
(Home with sick children)

Wednesday 17.02.2011: 0 hours
(Home with sick children)

Thursday 18.02.2011: 0 hours
(Home with sick children)

Friday 19.02.2011: 0 hours
(Home with sick children)

Saturday 20.02.2011: 0 hours
(Home with sick children)

Sunday 21.02.2011: 0 hours
(Home with sick children)

Week 8 23 hours

Monday 21.02.2011: 0 hours
(Home with sick children)

Tuesday 22.02.2011: 7 hours
Meeting 2 hours
Rendering testing/coding 5 hours

Wednesday 23.02.2011: 4 hours
Rendering testing/coding 4 hours

Thursday 24.02.2011: 8 hours
Rendering testing/coding 6 hours
Meeting Jayson(Update routine) 2 hours

Friday 25.02.2011: 3 hours
Rendering testing/coding 3 hours

May 27, 2011

Saturday 26.02.2011: 0 hours

Sunday 27.02.2011: 3 hours

Rendering testing/coding 2 hours

Research Update routine 1 hour

Week 9 20 hours

Monday 28.03.2011: 4 hours

Research update routine 2 hours

Scrum meeting 2 hours

Tuesday 01.03.2011: 6 hours

Scrum meeting 3 hours

Research updaterroutine 2 hours

Meeting Jayson/Simon UR 1 hour

Wednesday 02.03.2011: 3 hours

Research updaterroutine 3 hours

Thursday 03.03.2011: 6 hours

Testing updaterroutine 3 hours

coding updaterroutine 2 hours

Research updaterroutine 1 hour

Friday 04.03.2011: 0 hours

Saturday 05.03.2011: 0 hours

Sunday 06.03.2011: 1 hours

Coding updaterroutine 1 hour

Week 10 25 hours

Monday 07.03.2011: 4 hours

Coding updaterroutine 2 hours

Research updaterroutine 2 hours

Tuesday 08.03.2011: 6 hours

Coding updaterroutine 4 hours

May 27, 2011

Meeting updaterroutine 2 hours

Wednesday 09.03.2011: 5 hours
Coding updaterroutine 5 hours

Thursday 10.03.2011: 6 hours
Coding updaterroutine 6 hours

Friday 11.03.2011: 4 hours
Coding updaterroutine 4 hours

Saturday 12.03.2011: 0 hours

Sunday 13.03.2011: 0 hours

Week 11 19 hours

Monday 14.03.2011: 0 hours

Tuesday 15.03.2011: 5 hours
Coding updaterroutine 5 hours

Wednesday 16.03.2011: 5 hours
Coding updaterroutine 5 hours

Thursday 17.03.2011: 6 hours
Coding updaterroutine 6 hours

Friday 18.03.2011: 3 hours
Coding updaterroutine 3 hours

Saturday 19.03.2011: x hours

Sunday 20.03.2011: x hours

Week 12 25 hours

Monday 21.03.2011: 4 hours
Research Actionsystem 4 hours

Tuesday 22.03.2011: 6 hours
Research Actionsystem 2 hours

May 27, 2011

Design Actionsystem	4 hours
Wednesday 23.03.2011:	6 hours
Coding ActionSystem	6 hours
Thursday 24.03.2011:	5 hours
Coding ActionSystem	5 hours
Friday 25.03.2011:	4 hours
Coding ActionSystem	4 hours
Saturday 26.03.2011:	x hours
Sunday 27.03.2011:	x hours

Week 13 27 hours

Monday 28.03.2011:	6 hours
Coding ActionSystem	5 hours
Meeting	1 hour
Tuesday 29.03.2011:	7 hours
Coding ActionSystem	5 hours
Design ActionSystem	2 hours
Wednesday 30.03.2011:	4 hours
Design ActionSystem	4 hours
Thursday 31.03.2011:	5 hours
Coding ActionSystem	5 hours
Friday 01.04.2011:	5 hours
Coding ActionSystem	5 hours
Saturday 02.04.2011:	x hours
Sunday 03.04.2011:	x hours

WORKLOG ANDERS EINAR HILDEN

May 27, 2011

Week 1: 0 hours
Sick

Week 2: 0 hours
Sick

Week 3: **hours**
Monday 17.01.2011: hours

Tuesday 11.01.2011: hours

Wednesday 12.01.2011: hours
Started working on offsite GIT-hosting.

Thursday 13.01.2011: hours
Sendt public keys for group 1 hour
to GIT-host for access.
Meeting w. Jayson re. gameserver
Setup EGit, Eclipse, initialice repository
Research UDP connections pc/Android

Friday 14.01.2011: hours

Saturday 15.01.2011: 0 hours

Sunday 16.01.2011: 0 hours

Week 4
Monday 2011-01-25: **6 hours**
Digitalicing meeting notes from friday **3 hours**
Setting up a rudamentary webpage 2 hours

Thursdag 2011-01-26 hours

TODO:
SCRUM foredrag
GIT-testing (crashtest, etc)

Week 5
Tirsday 11-02-01
12-13: Eclipse configuration, setup etc.
bestemte oss for å gå for svn i stedet for å bruke masse tid på å lære oss git

13:30 -> udp hole punching, android udp test app

15:35 android udp test app complete.

15 februar:

09:00-12:00 sette opp eclipse, suberverison etc (utviklingmiljø) på hjemmemaskin

12:00-13:00: sette opp ny offsite server for hjemmetilgang til ragserver.php

13:00-15:00 login.php

16. februar

09:30 - xx:xx registrer.php

17. feb

16:45-19:00 -> se svn log

22:00 -> 01:30 -> se svn log

21. mai

Display health and mana of both players on screen

Create RegenController for regeneration mana of local player

22. mai

Include mainmenu in wizardwars

Rewrite mainmenu to accept games from server

Rewrite server

23. mai

Total rerwrite of-game part of server.

APPENDIX 3: STATUS REPORTS

24.05.2011

Hello, third and last status report of the bachelor project.

With 72 hours left of the project period, this is what we got:

- The game is almost done. We have a world, with some walls and boxes, and two players. The network part is what's left. The movement of the players are working, and now we are working on getting the spell system working correctly over network. The register/login user part is now connected to the game, and the server connects two players that are looking for an opponent.
- There are still some work left on the report. We need to tell about network in the design document. We also need to write a discussion and conclusion part.
- The code also needs some clean-up.

Although a little too much work still remains, we are very pleased with what we've done, and hopefully we will get everything done in time.

Jon S Lande, Ole Marius Kohmann, Anders Einar Hilden

04.05.2011

Hi, here's a second update on the progress of our bachelor project.

The project is due in three weeks, and this is what we have accomplished so far:

- The scene graph is now working as planned. We have a separate project that builds up a initial scene graph file (.sg) from game object files (collada, .dae). This will contain all the game objects we need.
- We have navigation icon for movement of the player
- We have some controllers for connecting and changing the nodes in the scene graph. This includes a translate controller, for movement of objects (players), rotate controller for rotation of objects, animation controller, for animation of player, and action controller, for controlling spells. Translation is done (we can move our own player around), the others need a little more work.
- A collision system, for detecting and handling collisions between objects. Supports sliding (e.g. player sliding against a wall)
- Sending/receiving simple packages over network between two phones works fine. We have made a system for creating network packages that contains everything we need to send/receive, but we haven't tested this yet. This is probably our biggest concern at the moment.
- We also have a system for casting spells. This is tested and seems to be working on its own, but for this to work over the network, we probably need to modify some of it. We have support for three different spell gestures on the screen.

The coding part of the project has taken more time than expected, so the report has not been prioritized yet. However, we have started on the report now, and will focus on this the rest of the project period along with getting the game done.

Since easter, Anders Einar is occupied with other things during the day, until after 25th of

May, so the group is a little amputated at the moment. He is, however, working in the afternoons and weekends.

Jon S Lande, Ole Marius Kohmann, Anders Einar Hilden

23.02.2011

Hi, here's a little status report on our bachelor project, three weeks into the development period.

Initially, the plan was to have scrum sprints on two weeks. However, the first sprint had to be extended with a week, since we didn't get any of the project parts done in time. We realized that because of our limited experience with game programming and Android, we don't know exactly how long time we need on each part. Therefore, from now on we will shorten our scrum sprints to **one** week.

According to the Gantt chart in the project plan, we were supposed to have finished the GUI, the object parser, server, scene graph, render routine and player by now. We saw early in the sprint that this was too much work.

The GUI is pretty much done. We have made a player registration screen, and a login screen on the phone. We also have a simple main menu. Login and registration are implemented on the server as well, using PHP with a MySQL database. The server also supports storing of players stats. We will need more functionality on the server (i.e. showing and connecting players), but there's no hurry with the rest of the server.

We also have two parsers for game objects, .obj files, and materials the objects use, .mtl files. These are files outputted from 3D Studio Max, which we will use to model game objects. We are going to create nodes in the scene graph from the .obj files, as soon as the scene graph design is ready. We are currently testing the scene graph, so this is coming along well. The main tasks for us for this weeks sprint us to get the scene graph done, and the render routine.

The last week of the sprint got a little amputated, since Ole M had to stay at home with sick kids. Still, we are pretty happy with what we have accomplished so far.

Ole Marius Kohmann, Anders Einar Hilden, Jon Lande

APPENDIX 4: SCRUM MEETINGS

SPRINT 8 (05.04.2011 - 15.04.2011)

Sprint planning meeting

For this sprint:

- Finish collision system
- Finish player movement
- Get opponent players movement over network
- Make an animation system for player movement

Sprint review meeting

Sprint retrospective meeting

Sprint 7 (29.03.2011 - 04.04.2011)

Sprint planning meeting

For this sprint:

- Collision handling - handle collisions between objects

Sprint review meeting

The collision system should now handle the collisions correctly, though we haven't tested it with real objects yet, only in a test environment.

Sprint retrospective meeting

SPRINT 6 (22.03.2011 - 28.03.2011)

Sprint planning meeting

For this sprint:

- Continue on collision detection

Sprint review meeting

We now have a system that (in theory) detects collisions. We use rectangled bounding boxes around objects for detecting collisions.

Sprint retrospective meeting

SPRINT 5 (15.03.2011 - 21.03.2011)

Sprint planning meeting

For this sprint:

- Start on collision detection.
- translate controller.
- human/player/opponent.
- onscreen controller.
- Gestures research.

Sprint review meeting

This sprint we had a meeting with the supervisor, who was pleased with our progress. We got started on the collision detection system, for detecting and handling collision between objects in the game.

Sprint retrospective meeting

SPRINT 4 (08.03.2011 - 14.03.2011)

Sprint planning meeting

For this sprint:

- Finish website
- Make collada file parser
- Continue on HUD
- Continue on update/controller routine

Sprint review meeting

We have moved from a .obj file parser and made a xml/Collada parser. This was necessary due to unforeseen constraints in the .obj file format.

Updateroutine is at stage 1 and tested with a rotatecontroller.

HUD is visible and Anders Einar will implement features for a onscreen controller next sprint.

Sprint retrospective meeting

SPRINT 3 (01.03.2011 - 07.03.2011)

Sprint planning meeting

For this sprint:

- We are going to finish the object parsing, so objects are displayed correctly on screen.
- Start on HUD (Heads up display) - Ingame GUI
- Get texture mapping working, so the objects textures are loaded automatically.
- Start on update routine / controllers
- Finish website

Sprint review meeting

We managed to finish the object parsing, so the objects are displayed properly. However, the textures are not mapped correctly, so this still needs some debugging. We started on the HUD/Ingame GUI, and this looks good so far. We didn't have time to finish the website, so this will be the first priority next sprint.

Sprint retrospective meeting

This sprint we worked a little closer than before, helping each other with coding, instead of only working on our own parts of the project. We feel this was helpful.

The planning for this sprint could have been better. We could have defined the goals of the sprint better.

SPRINT 2 (22.01.2011 - 28.02.2011)

Sprint planning meeting

This sprint we will focus on getting the scene graph code done, so we can try to write it to file. From there we are going to read these files and build up the graph from file(s) every time we need it.

Tasks:

Scenegraph I/O

Renderer

Sprint review meeting

This sprint we all worked on the rendering routine for a game object. The goal was to parse an .obj-file with game objects, exported from a 3D modelling program, correctly, and then put these objects into nodes in our scene graph. From there we will render the objects to screen. However, we didn't manage to get this working correctly, the objects don't look as expected on screen. The scenegraph and rendering code has been tested ok, so it looks like there is some incorrect reading/saving of data from the .obj files that is causing it. This should be a small issue to fix.

Sprint retrospective meeting

This sprint was used to try to finish the object parsing and rendering, which was the unfinished parts from last sprint. Even though we didn't quite finish it, we're almost there. It has been a lot of trying and failing, so maybe we could have planned the code better. However, we are learning a lot.

SPRINT 1 (31.01.2011 - 21.02.2011)

Sprint planning meeting

For this first sprint:

-

Sprint review meeting

This sprint had to be extended with a week, since we didn't get any of the project parts done in time. We realize that because of our limited experience with game programming and Android, we don't know exactly how long time we need on each part. Therefore, from now on we will shorten our scrum sprints to **one** week.

According to the Gantt chart in the project plan, we were supposed to have finished the Graphical User Interface, the object parser, server, scene graph, render routine and player by now. We saw early in the sprint that this was too much work. The GUI and object parser is almost finished, and the user part of the server is done. There's no hurry with the rest of the server. We need some more time to work on the scene graph and render routine.

The last week of the sprint got a little amputated, since Ole M had to stay at home with sick kids. Still, we are pretty happy with what we have accomplished so far.

Sprint retrospective meeting

What went well:

- We have learned a lot already
- Although we are behind our schedule, we feel that we've come a pretty long way.
- We work pretty good together

What could have been done better:

- Planning, especially of the server
- Better documentation of all the choices we make
- Each member could be more open about what he is working on, so everyone knows what's going on.

APPENDIX 5: DAILY SCRUMS

DAILY SCRUM MEETING, 2011-03-08

OMK

What did you do yesterday?

Coded update routine, should work in theory now. Need to implement some more stuff to test properly.

More research on the same topic.

What will you do today?

Scrum planning meeting.

Implement more code for the update routine.

Are there any impediments in your way?

no.

JSL

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

AEH

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

DAILY SCRUM MEETING, 2011-03-01

OMK

What did you do yesterday?

Researched on Update routine method.

Scrum meeting

What will you do today?

Scrum planning meeting.

Continue to research the update routine method and figure out how to do it.

Talk to Simon and Jayson about my ideas for the update routine.

Are there any impediments in your way?

no.

JSL

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

AEH

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

DAILY SCRUM MEETING, 2011-02-25

OMK

What did you do yesterday?

Rendering code(texture node)

What will you do today?

Same

Are there any impediments in your way?

Not really

JSL

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

AEH

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

DAILY SCRUM MEETING, 2011-02-24

OMK

What did you do yesterday?

Rendering, (geometry node)

What will you do today?

Rendering, more nodes

Are there any impediments in your way?

not really

JSL

May 27, 2011

What did you do yesterday?

Working on adding game objects to the initial scene graph, from .obj files

What will you do today?

The same as yesterday

Are there any impediments in your way?

Hopefully not

AEH

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

DAILY SCRUM MEETING, 2011-02-23

OMK

What did you do yesterday?

What will you do today?

Rendering

Are there any impediments in your way?

not really

JSL

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

AEH

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

DAILY SCRUM MEETING, 2011-02-16

OMK

What did you do yesterday?

Nothing(Home with sick child)

What will you do today?

SceneGraph testing

Are there any impediments in your way?

Not really, fix code if its not working...

JSL

What did you do yesterday?

Finished a basic login screen.

What will you do today?

Continue on the player registration.

Are there any impediments in your way?

Need Anders Einar to have the server ready in order to test registration.

AEH

What did you do yesterday?

Login part of server

What will you do today?

Player registration part of server

Are there any impediments in your way?

Daily scrum meeting, 2011-02-10

OMK

What did you do yesterday?

Scenegraph traversing

What will you do today?

SceneGraph traversing

Are there any impediments in your way?

No

JSL

What did you do yesterday?

Started on the gui. Planned how it should look, made a login screen.Planned the database

What will you do today?

More Gui and database + login functionality.

Are there any impediments in your way?

No

AEH

What did you do yesterday?

Fant & tilpasset en reliable, high-volume server. Skrev en echo-server. Begynte på en RagHandeler, testet å sende javaobjecter via serveren

What will you do today?

Sende javaobjekter til en androidtelefon. Deretter starte på server-api'et.

May 27, 2011

Are there any impediments in your way?

Trenger kanskje litt hjelp og input ang. database og gui fra JSL

Daily scrum meeting, 2011-02-09

OMK

What did you do yesterday?

SG tree build up

What will you do today?

SG tree build up

Are there any impediments in your way?

No

JSL

What did you do yesterday?

Researched sounds in Android

What will you do today?

Start on the GUI

Are there any impediments in your way?

No

AEH

What did you do yesterday?

Researched server

What will you do today?

Write server, help JSL w. GUI, working on serverrequests in co-op JSL

Are there any impediments in your way?

No

DAILY SCRUM MEETING, 2011-02-08

OMK

What did you do yesterday?

SG traversing methods research

What will you do today?

SG tree build up

Are there any impediments in your way?

Nothing yet, but will need a testfile for tree structure eventually.

JSL

What did you do yesterday?

materialfile parser, reading OpenGL

What will you do today?

Sound

Are there any impediments in your way?

no

AEH

What did you do yesterday?

Sick(friday, saturday,sunday,monday)

What will you do today?

Serverstuff

Are there any impediments in your way?

Nothing major.

DAILY SCRUM MEETING, 2011-02-03

OMK

What did you do yesterday?

SceneGraph (Nodes & started on traversing)

What will you do today?

SceneGraph(More Traversing)

Are there any impediments in your way?

Not really.

JSL

What did you do yesterday?

Continued work on the .obj parser. Should be working, Have been committed.

What will you do today?

Starting on the .mtl parser.

Are there any impediments in your way?

no.

AEH

What did you do yesterday?

Meeting with IT about testing. Two meetings with Jayson. Research.

What will you do today?

Starting on servercode.

Are there any impediments in your way?

no.

DAILY SCRUM MEETING, 2011-02-02

OMK

What did you do yesterday?

Worked with scenegraph, helped, JSL. Nodes in SC. Research. Planning.

What will you do today?

The same. Continuing on nodes. Continue helping JSL.

Are there any impediments in your way?

Not really, just needs work.

JSL

What did you do yesterday?

Learning from OMK. Worked on parser. Read almost everything, and group it into lists.

May 27, 2011

What will you do today?

Continue on parser. Syncing with OMK nodes. Starting materials-file parser.

Are there any impediments in your way?

Needs reference implementation from OMK regarding Nodes before i can continue.

AEH

What did you do yesterday?

Researching NAT/PAT. Got PoC(Proof of Concept) client on the phone, can send and receive UDP packets.

Came across problems with phone to phone communication, did more research on the topic(STUN/ICE).

What will you do today?

Meeting with IT-department @ school regarding networks theory, and planned testing.

Are there any impediments in your way?

Small issues, need to discuss with group to find a good solution.

APPENDIX 6: FIRST TECHNICAL DESCRIPTION

What we need

Tools	Engine	Game
<ul style="list-style-type: none"> - Obj2SG - Servertool - Result storage - Userstats storage 	<ul style="list-style-type: none"> - SceneGraph - Collision - Physics - GUI - EventHandler - Renderroutine - Network - Movement - Gestures - Item system - Spell system - Talenttree system 	<ul style="list-style-type: none"> - Player - Item spec - Spell spec - Talenttree spec - Talent logic - Level win/lose logic

OBJ2SG

Needs:

- .mat reader/loader
- .obj reader/loader
- texture reader (->OGL)

Purpose:

1. To read a obj file & create a Scene Graph file from it. The SG file is then loaded into the game
2. Convert obj files to dynamic object files (Like player models)

SERVER

Packets:

- Movements
- Actions

TCP-queries:

- Aftermatch data
- Prematch data

Sane-packet-check

Matchmaking

ENGINE

SceneGraph:

Classes:

- Node (abstract)
 - Virtual function execute();
- Translate node (extends Node)
- Rotate node "-----"
- Scale node "-----"
- Model node "-----"
- SceneGraph
- Traverser
 - Depth first
 - Performs Execute() on Current node if dirtyFlag i marked
 - Unflags dirty flags

Collision/Physics:

- line-circle collision
- Ray intersection test
- Simple gravitational physics

APPENDIX 7: PROJECT PLAN (PRE PROJECT REPORT)

Project Plan

For Bachelor Thesis 2011

RAG: Really Awesome Game

Ole Marius Kohmann - 071192

Anders Einar Hilden - 080207

Jon Sætheren Lande - 080822

All participants are from 08HBINDA

5 TABLE OF CONTENTS□

[Table of contents](#)

[1. GOALS AND BOUNDARIES](#)

[1.1 Background](#)

[1.2. Goals](#)

[1.3. Boundaries](#)

[2. SCOPE](#)

[2.1. Project description](#)

[2.2. Scope](#)

[3. Project organization](#)

[3.1 Employer and supervisor](#)

[3.2. Responsibilities and roles](#)

[3.3. Group rules and routines](#)

[3.3.1 Group rules](#)

[3.3.2 Routines](#)

[3.4 Resources](#)

[4. Planning, meetings and reporting](#)

[4.1. System development model](#)

[4.2. Meetings](#)

[4.3 Status reports](#)

[5. Quality assurance](#)

[5.1 Testing](#)

[5.2 Code meeting](#)

□

6 1. GOALS AND BOUNDARIES

1.1 BACKGROUND

Wanting to challenge ourselves we have decided to develop a game as our final project for our bachelor degree. Writing a game is a complicated task, requiring knowledge from many of the courses we have had already, mentioning a few: Programming, algorithms, operating systems, computer architecture, WWW-technology, Program development, systems engineering, math, physics, etc.

1.2. GOALS

Our main goal is to develop an entertaining multi-player mobile game for the android platform. We want to reach out to as many people as possible, and the android platform is the fastest growing mobile platform. The game needs to have good performance, and be user friendly.

Another important goal for the project is the learning aspect. We are excited about learning game programming, programming for mobile units/Android, as well as improving our overall programming skills.

1.3. BOUNDARIES

The project is due 25.05.2011. This gives us almost five months to develop the game. Out of the three persons on the group, Ole Marius is the only one with experience from game development, so we will need to use some time on learning game technology.

We are going to develop the game for the Android platform, specifically Android 2.2. Although Android 2.3 is the latest version at the moment, we believe we will reach out to more people by supporting 2.2. Because of the time limit, we won't be able to support other versions or platforms.

We are limiting ourselves to developing the game first and foremost for the android phone HTC Desire. This is one of the most popular android phones on the Norwegian market (http://www.amobil.no/artikler/dette_er_favorittmobilene_til_netcoms_kunder/80762 & http://www.amobil.no/artikler/her_er_de_25_mest_populaere_mobilene/80621/2)

7 2. SCOPE

2.1. PROJECT DESCRIPTION

A multi-player game that allows players to play competitive against each other. We will use Android for the client and Java for the server side. The game's setting is a fight/battle in a arena between two players. Winner gets a reward, loser don't. After a match, the player earns experience, which allows him to get points to use to develop/improve his character. The player can get a limited amount of points and there are more abilities than there are points. This allows for variation in player style, which gives a more interesting gameplay.

The game will have a user interface where players can get random suggestions for opponents and accept or reject these.

The player will be controlled with the trackball/optical sensor for movement and using gestures on the screen for selecting and using spells/skills.

2.2. SCOPE

We want the game to be technically functional at the end of the project period. This means that we have to keep the artwork to a minimum. All gameplay features described in the description should be implemented and functional.

Both client and server is going to be operational at the end of the project.

8 3. PROJECT ORGANIZATION

3.1 EMPLOYER AND SUPERVISOR

The projects employer is Associate Professor Simon McCallum at the Game Technology Lab, Gjøvik University College.

Our supervisor is Assistant Professor Øyvind Kolloen.

3.2. RESPONSIBILITIES AND ROLES

Ole Marius Kohmann: Group Leader, Lead Designer

Jon Sætheren Lande: Project Manager

Anders Einar Hilden: Technical Supervisor

Scrum roles:

- Scrum master: Anders Einar
- Product owner: Ole Marius

- Ole Marius is responsible for the idea.
- Jon is responsible for the project reports.
- Anders Einar is responsible for the website and the version control.

This only shows who's responsible for getting the different parts of the project done. We will obviously be cooperating closely on most parts.

3.3. GROUP RULES AND ROUTINES

3.3.1 GROUP RULES

- All of the group members have to log all of their work - what they do, and time spent.
- Whenever there's a conflict in the group, the general rule is that the majority of the group rules.

3.3.2 ROUTINES

Work hours are Tuesday - Friday 09.00- 15.00. If nothing else is agreed, we show up at our bachelor room 09.00 these days.

Broken rules / routines are logged. If this log grows to a substantial size, this log will be added to the final log as an appendix. Any serious offenses will be discussed as a group along with the supervisor.

3.4 RESOURCES

In order to do/test the project, we need some equipment. We have borrowed a desktop computer from the IT department at school. We will also be needing some Android phones/tablets that the Game Tech Lab will provide us. It's important to test the software on different kinds of hardware that runs Android.

Our project employer, Simon McCallum, will, along with Jayson Mackie at the Game Tech Lab, function as technical advisers during our project. They both have a lot of game programming experience.

9 4. PLANNING, MEETINGS AND REPORTING

4.1. SYSTEM DEVELOPMENT MODEL

When we looked at different models we quickly agreed that we wanted to use an agile method as our development model. An alternative would be to choose the waterfall method, but this method is very static and requires a lot of planning before development. This might be good for larger developers, but for a bachelor project it is better, in our opinion, to have a method that allows a more dynamic way of working. Agile methods opens up for this.

When looking at the different methods we narrowed it down to 3 candidate methods; RUP, Scrum and Extreme Programming(XP).

RUP is a very advanced method, with alot of tools and subsystems that can be used to work for smaller teams. But, all in all, RUP is best used in large teams and big projects.

XP is the opposite, it favors creation over documentation and also requires a very active project owner. XP could fit, if working extra on documentation, but project owner is not available at all times.

Conclusion was that even though XP could be used, it would not be proper XP, so we chose to look elsewhere.

Scrum is a lot easier than RUP to both use and understand, as well as having better documentation methods than what XP has. We also believe that reporting to the project owner after each sprint is better suited for us and the owner, than having the owner physically represented in the development team at all times as XP requires.

Scrum uses scrum sprints to split up the development. at the start of each sprint you decide what parts you should work on for this sprint, by selecting tasks from a previously created backlog that contains all the tasks needed to finish the project. When a sprint is finished, the group will have a set of functional parts of the project that can be tested and used further. Working like this also opens for adding new tasks to the backlog as you progress or put non-

finished tasks back into the backlog if deadline was not met.

The group has agreed together on using Scrum as our System development model.

4.2. MEETINGS

- Daily scrums. Each member of the development team takes 5 minutes to sit down and answer 3 questions:
 - What have you done since yesterday?
 - What are you planning to do today?
 - Do you have any problems preventing you from accomplishing your goal?
- Midweek meeting: Thursdays. Quick meeting to look and review each-others code together.
- Sprint planning meeting. The first Tuesday in all sprints, we will have a meeting to decide what to do in the coming sprint.
- Sprint review meeting. Last day in each sprint we will have a meeting to demonstrate & review the finished parts from our sprint. Unfinished parts will not be demonstrated.
- Sprint retrospective. The Monday after a sprint, we will have a meeting where we discuss our past sprint. The task each member of the project has in this meeting is to answer two questions: What went well in the last sprint, and what could be improved in the next sprint?
- At the start of each scrum iteration, we will have a meeting with our supervisor. Here we will discuss the project process, management and documentation.

4.3 STATUS REPORTS

Each week we will make a short status report, describing the progress of our project. This will be sent to our employer and our supervisor, so they easily can follow our progress.

10 5. QUALITY ASSURANCE

5.1 TESTING

Each task in a sprint has a test written for it that the task has to complete in order for the task to be accepted as complete.

In the finalising part of our project, we will use extra time testing, both internally and externally, to identify problem areas and fix bugs.

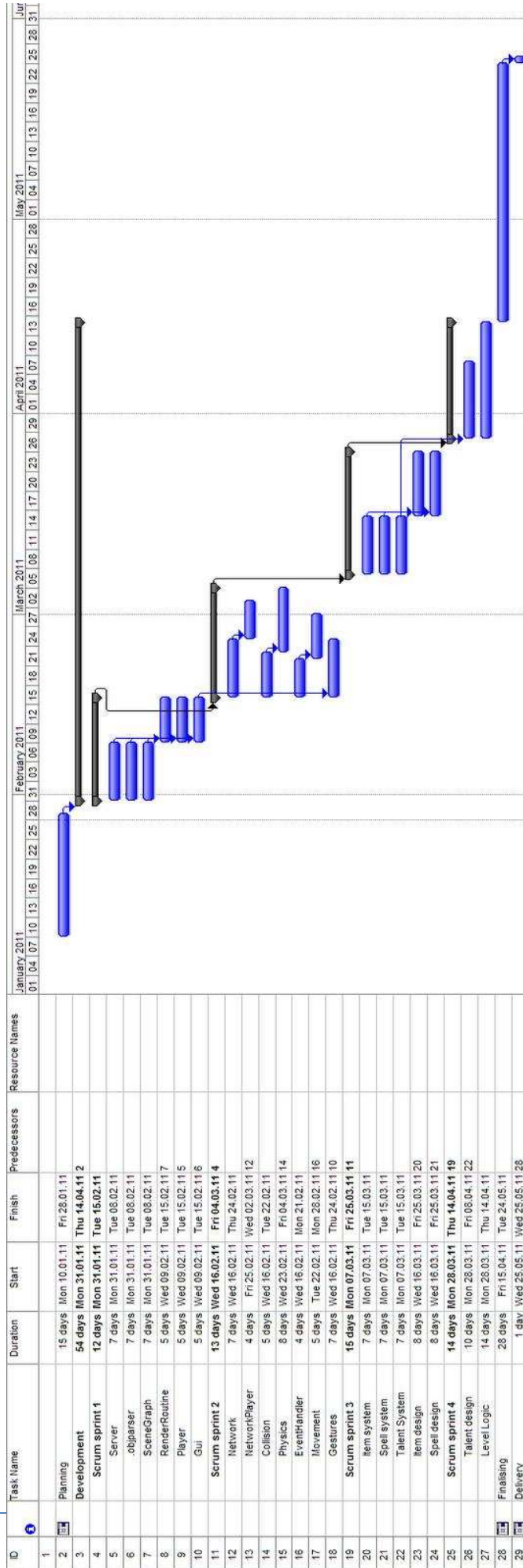
5.2 CODE MEETING

Once a week we will have a group meeting where we read through each others code. This way we assure that the quality of the code is maintained.

6. GANTT CHART

Since we are using Scrum as our development model, the development period of the project will be divided into sprints. We set one sprint to be two weeks. This gives us four sprints total. At the end of each sprint we will have a working product.

Below is our Gantt chart, showing the progress plan for the project.



APPENDIX 8

Overview of the code delivered

- **Wizard wars.zip**
 - Our complete code-three for our Android application
 - The internal JavaDocs for the sourcecode
 - All graphics, and a playable gameworld
- **Web Server.zip**
 - Our complete code-three for game server
 - A SQL file for recreating our server database w. three users
- **createSG.zip**
 - Our complete code-three for the tool that parses the game objects, and builds up a file with an initial scene graph
 - This project requires the Android.jar and Wizard Wars project on it's classpath to compile
- **Wizard Wars.apk**
 - A Android APK file for installing the game on Android devices



HØGSKOLEN I GJØVIK

PROSJEKTAFTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

JASON MACKIE

(oppdragsgiver), og

Ole Marius Kohmann, Anders Einar Hilden og
Jon Sætheren Lande

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 01.01.11 til 25/5-11.

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG yter veiledning.

Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra HiG å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
- Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra HiG. Studentene dekker utgifter for trykking og ferdigstilling av den skriftlige besvarelsen vedrørende prosjektet.
 - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.
3. HiG står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.
4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved HiG og/eller studenter har interesser.

Besvarer med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.
6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.
7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.
8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.
9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene.

Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale, skjer dette uten HiG som partner.

9a. Prosjektets intellektuelle rettigheter tilhører gruppens medlemmer(Ole Marius Kohmann, Jon Lande og Anders Einar Hilden) både før og etter prosjektets avslutning. Eventuelle beslutninger om videre bruk av IP må avtales mellom de tre gruppemedlemmene.

10. Når HiG også opptrer som oppdragsgiver trer HiG inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): Dywind Kolloen

Oppdragsgivers kontaktperson (navn): JASON MAREKIE

Student(er) (signatur): Jon S. Lande dato 31.01.11
Anders Einar Hilden dato 11-01-31
Ole Marius Kohmann dato 31.01.11
_____ dato _____

Oppdragsgiver (signatur): [Signature] dato 31/01/11

IMT Dekan/prodekan (signatur): [Signature] dato 2.2.2011