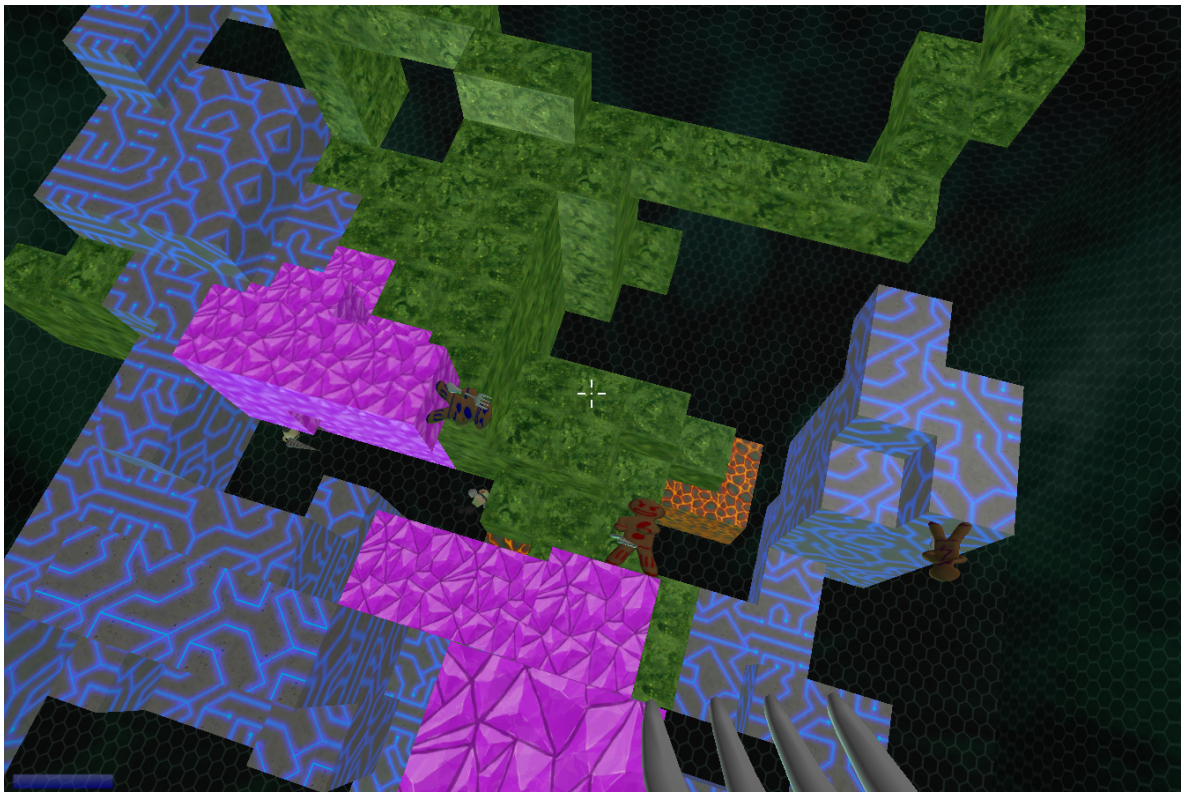
Surreal Gravity

Group 2



Contents

1 Introduction **2**

2 The Team **3**

3 Target Audience **4**

4 Platform & Controls **4**

5 Story, characters and setting of the game **5**

6 Technical components **5**
6.1 Navigation 5
6.2 Gravity-switch system 5
6.3 Multiplayer connection 7
6.4 Network Synchronization 7
6.5 Web Server 7
6.6 Account Management 8
6.7 Menu 8
6.8 Level creation 9
6.9 Robot 9

7 Code quality **10**

8 Art design **11**
8.1 Models 11
8.2 Audio 11
8.3 Interface 13
8.4 Textures 13

9 Process **15**

10 Conclusion **16**

1 Introduction

This is a report of our game Surreal Gravity. We will cover the components that make up the game, and some general topics regarding the process of creating our game.

2 The Team

Game Designer
Kees Kroep 4246373
Electrical Engineering
k.kroep1@gmail.com

Lead Programmer
Roberto Mol 4206886
Electrical Engineering
roberto_mol@hotmail.com

Lead Artist
Kwok Hou Man 4225767
Civil Engineering
kwokhou.nl@gmail.com

Lead sound design
Ayyoeb Ichaarine 4224922
Industrial Design Engineering
a.ichaarine@gmail.com

World designer
Coen Berssenbrugge 4243838
Mechanical Engineering
C.W.J.Berssenbrugge@student.tudelft.nl

Producer
Steven van der helm 4221362
Aerospace Engineering
S.vanderHelm@student.tudelft.nl

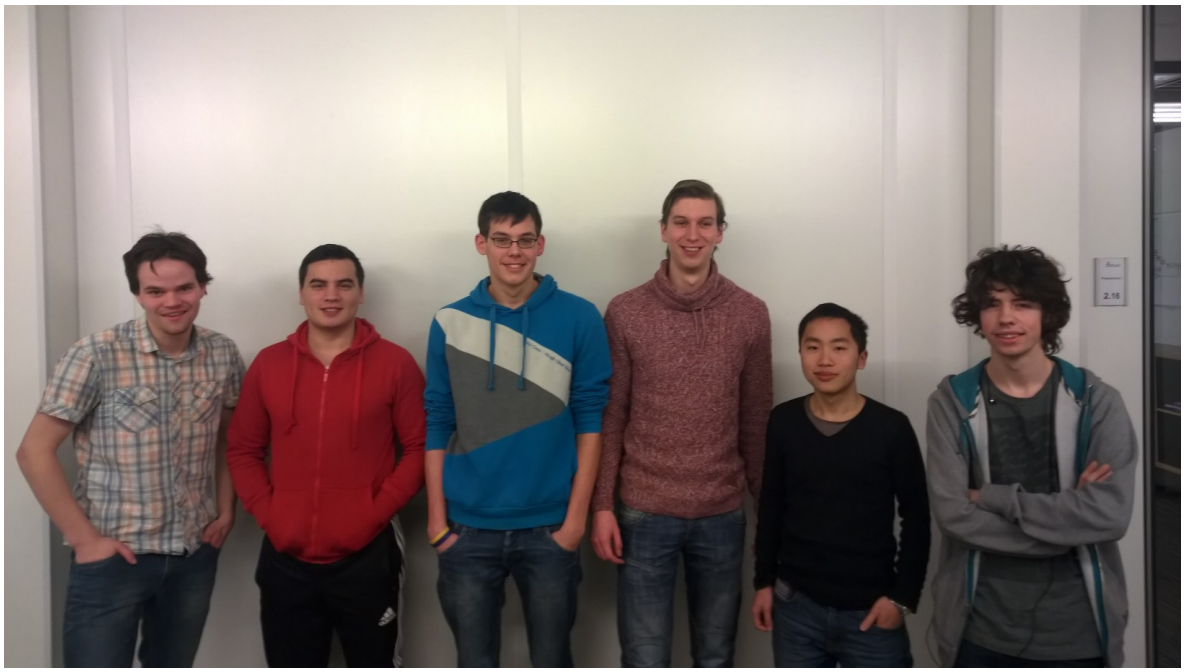


Figure 1: Group picture

3 Target Audience

The target audience is the PC Gamers who like to play fast paced multiplayer games. We believe that our game will offer an appealing challenge to those players. Concerning casual gamers; we have tried to make the game as accessible as possible, but the core mechanic of our game needs some time to get used to. We believe that messing with gravity this way has never been implemented this way.

4 Platform & Controls

The platform is PC (keyboard and mouse). The controls mimic the controls of a standard FPS game:

- WASD keys to move around in the horizontal plane
- Space bar to jump
- Mouse movement to aim your weapon and look around.
- Left mouse button to poke with your fork or shoot a death laser (depending on the chosen game mode)
- Right mouse button to fire a gravity laser that changes your gravity, depending on the normal vector of the surface it hits
- Tab to bring up the score screen
- C to lock your cursor and to make it invisible.

- ESC to return to the menu

5 Story, characters and setting of the game

Our game doesn't have a specific background story. The game has an intentional absurd feel to it and features both gingerbread men as well as futuristic men.

6 Technical components

In this section each component will be touched upon and a brief explanation will be provided.

6.1 Navigation

The navigation is essential to a good game. The navigation of the player is not trivial in our game, because there is no up or down. The player can switch gravity as he pleases (see next chapter). We decided to stick to standard FPS controls. When the player switches gravity, the player and camera will simply flip over. We have tried to implement other solutions, and did some testing on it, but we eventually decided to keep it this way. To help casual/new players we have decided to stick to the standard FPS controls.

We had a problem with walking. Whenever the player stopped walking, he got pushed back a bit. It turned out that forces and velocity's where used alongside each other, which is handled poorly by Unity.

We also feature a no-jump mode that really tests the skill of players to exploit gravity.(Kees)

6.2 Gravity-switch system

The core feature of our game is the ability to switch gravity. A player can shoot at a surface, and the normal vector of the surface becomes the new "up" vector for that player. This results in two things: It provides the player with a brand new way of navigating and is also aesthetically interesting; you will see players run on walls and ceilings as in an Escher painting.

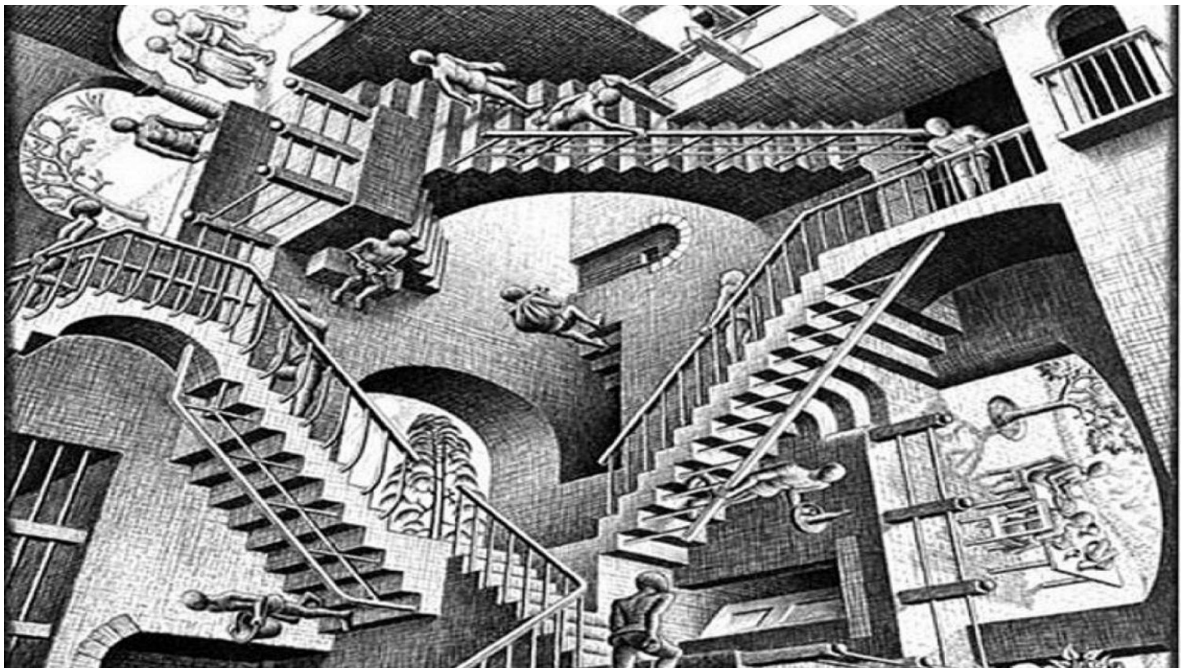


Figure 2: Famous Escher drawing

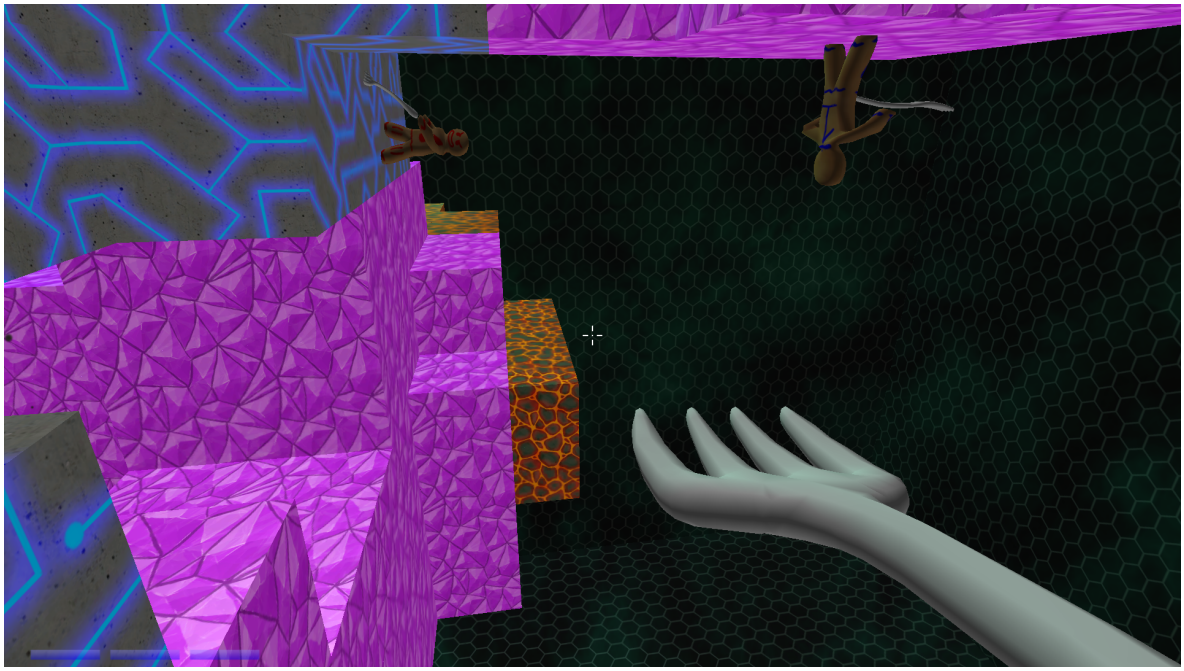


Figure 3: gravity shift effect

The implementation of this feature was not trivial; we iterated on it a lot. We went from a physical bullet to a raytrace and experimented with different transition methods. The main concern is the loss

of orientation. We have eventually chosen a system that tries to predict the direction you want to go and make the player face that direction. This is implemented primarily with vector operations. (Kees)

6.3 Multiplayer connection

The multiplayer part of our game is without doubt the biggest challenge.

The first issue we had was to connect clients to a host. Luckily Unity has decent support. First, the host needs to present itself to the clients. We do this by registering the host on the Unity Masterserver with a specific gametype. Clients can search the Unity Masterserver for hosts with this gametype. The players can then connect to one of the found hosts. We have put a lot of work into hiding the complexity of multiplayer to the players. We modified the game so it supports online as well as lan multiplayer. (Roberto)

6.4 Network Synchronization

When all the players are connected to the same host the real problems start. The game should run smoothly and everything should be synchronized on both the host and the clients. This means that almost all components of the game should have interaction with network things. You can think of scores, player names, kills, locations, rotations, weapons, levels, animations etc.

The problem with this is that networking only allows for limited transfer of data. A single quaternion is the most complex data type supported. We had to work with strings to send information about the level layout (which is procedurally generated), robots, and player information.

A second problem is lag and latency. The positions and rotations do not get an update instantaneously. This information updates approximately every 0.2 seconds. This means that other players move around in 5 frames per second, which is unacceptable in a fast paced game. Additional smoothing with lerps is applied to solve this problem. (Roberto - programming; Kees - programming)

6.5 Web Server

One of the main tasks is to implement a webserver that logs relevant information. The webserver has been built using node.js.

The webserver has to manage account registration and authentication. This was done via simple url encoding and communication of the usernames and passwords from unity to the webserver happens via these predefined URLs.

An issue that has been disregarded with respect to registration and authentication is the issue of security. The webserver has been protected against SQL injection into the database (all SQL characters that user may input are escaped), but usernames and passwords are sent via simple HTTP and are unencoded both in the URL and in the database. It was decided not to invest too much time into this issue because the accounts that are created for our games do not store any personal or private information about our users anyway. Instead we just advice users to use harmless passwords.

The second task is managing the database that is behind our game. This is a MySQL based relational database. The server is responsible for adding, updating and reading information from the database and is also responsible for making sure the game can communicate with the data present in the database. This has been done by defining predefined URLs that the game can access. These URLs will correspond to queries that the server will now run on the database, and the results will be returned

to the URL.

The final task that the webserver is responsible for is printing out the contents of the database to a web page. This has been done using HTML tables that correspond in a one to one fashion with the contents of the database.

We decided to visualize data on of the database in the game itself. The stats tab provides the player with useful and competitive information See figure 4.(Steven - all)

Figure 4: Ingame Stats

6.6 Account Management

In order to be able to store accounts, an Account class has been implemented.

Initially, before the web server was running, players could register an account (and log-in) with help of a text-file.

Later, we decided to keep the option of logging in locally. We are not able to maintain the Web Server forever, and we want people to still be able to play our game. We feel obliged to recommend you the following: don't use a password which is already used elsewhere. (Roberto - Programming)

6.7 Menu

Initially, we didn't implement the new Unity UI for our menu. Eventually, we ported our menu to the new UI. We have made use of buttons, texts, inputfields and sliders. Menu tabs consists of game objects and can be turned on or off. (Roberto - programming and port to Unity UI; Kees - initial design and programming; Ayyoeb - conceptual design)

Figure 5: Main menu

6.8 Level creation

The concept of our game is very suitable for a procedurally generated level. We decided that it would be good to use cubes exclusively, because these correspond perfectly with the six directions of gravity. These cubes are also very suitable for procedural level generation.

The level generator defines a three dimensional matrix that is initially set to only 0's. The generator will pick a predefined number of positions in the matrix and fill these positions with a random number ranging from 1 to 4, where each number corresponds to a type of building block that is used in the level. From these positions, the generator will continuously expand outwards, every time filling a random position next to the old position with the same number as the old position (1 to 4). This is a fairly random process and the islands will also look very random if they would be drawn into world space directly.

To remove some of this randomness, the second step of the level generator is to apply a smoothing filter over the matrix. This will remove unconnected paths from the level and make sure that paths that are almost connected will be fully connected. The result of this filter is a more coherent level.

The final step of the level generator is to go over the defined matrix and for every number greater than 0 the generator will instantiate a building block in world space, at a location corresponding to the index in the matrix. The level is now completed and ready to be played. (Steven - programming)

6.9 Robot

The main form of artificial intelligence in our game is the robots that fly around the level. Their task is to select blocks in the level, fly towards these blocks and destroy them. This process happens in several steps.

The first step in the process is to select appropriate blocks. The selection process of the robot is not entirely random. The robot will primarily select edges of the islands of blocks to destroy, because these will not significantly alter the smoothness of the level when destroyed.

After selecting a block, the next objective is to have the robot find a path towards the selected block. This is done using an A* algorithm. The level generator already has a matrix available, which can directly be used with the A* algorithm. The algorithm will traverse the matrix towards the target position. The heuristic function used in the A* algorithm is the horizontal plus vertical distance from the start point to the end point, as if there were no blocks obstructing the robot's path. When the end point is reached, a function will traverse back the path and store an array of consecutive nodes as the robot's path.

The final step for the robot is to take this path and fly towards the end point. This was done by using a combination of linear and spherical lerps, where the spherical lerps are used whenever the robot needs to turn. The robot will traverse the given path until it reaches the target cube. It will turn towards the cube and destroy it after a set time has passed. When the cube is destroyed, a smoke particle effect will be triggered to complete its destruction. The robot will then start all over again. (Steven - AI programming)

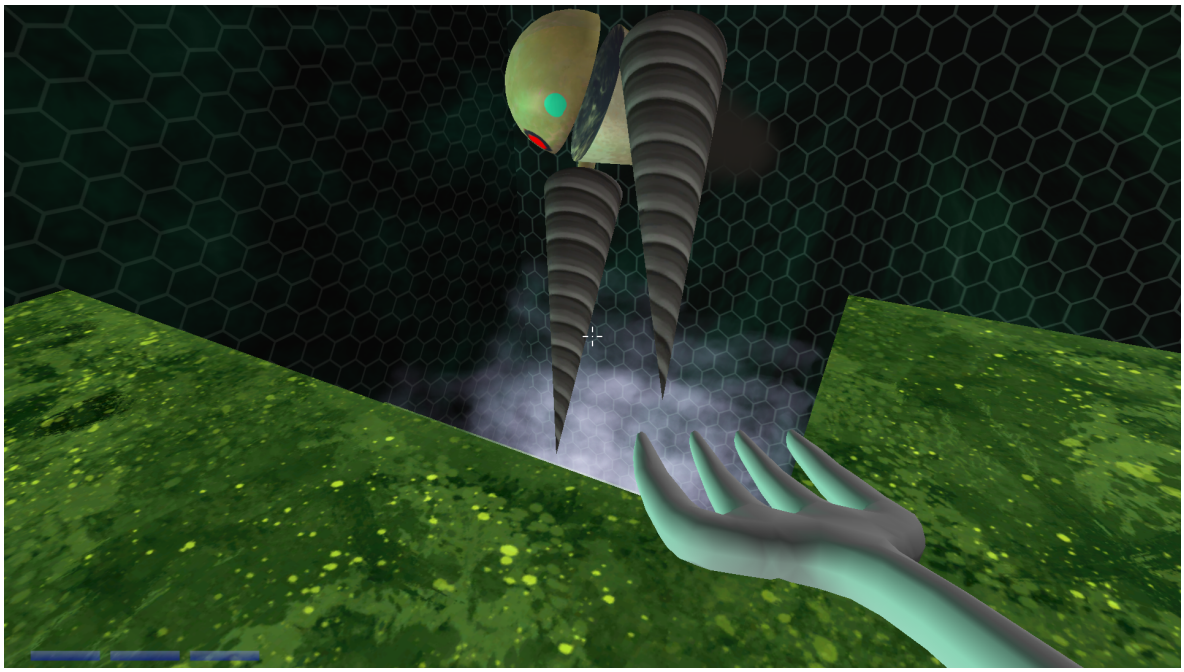


Figure 6: Robot in action

7 Code quality

Code quality has not been very consistent. There is is a big difference in readability across scripts. The worst readable script is the playercontroller.cs. This is the main controller of the players and holds a lot of functionality implemented by all members of our group. The other scripts are quite alright. In

the end, we put a bit of effort to make the scripts more readable.
We haven't had much issues regarding readability during our project.

8 Art design

8.1 Models

- Characters (normal and gingerbread)
- Gun (and bullet)
- Fork
- Robot

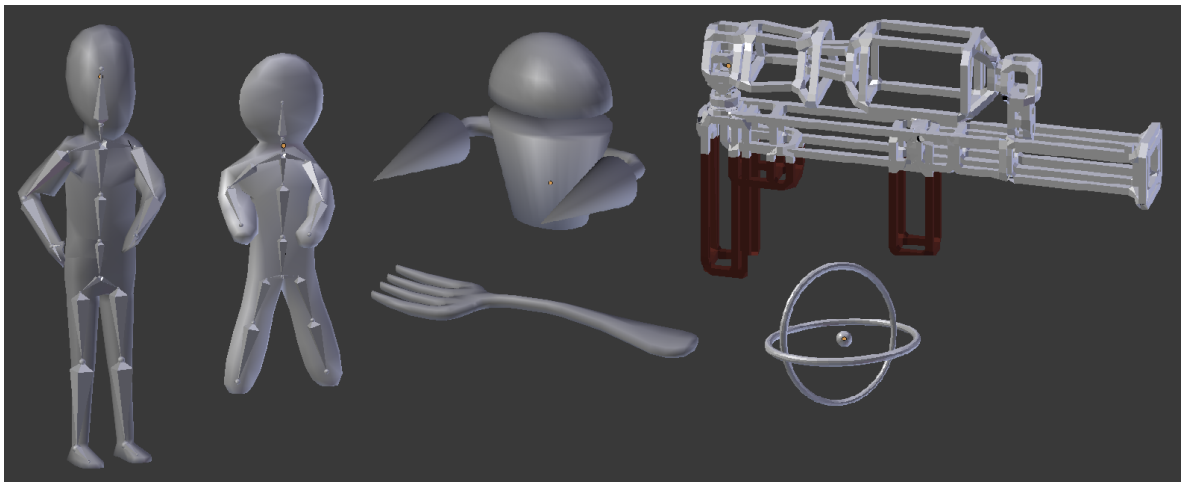


Figure 7: The models in game

These models are all selfmade in blender. We made two different characters to go with the two gamemodes. The normal character is a person made for the gunmode and we thought it would be fun to create a funnier mode and this happens to be the gingerbread with a fork. The robot is used to destroy blocks in the level, this is done by its drills which are his hands. The idle, walk and jump animations were created for both characters; the stab and swing animations were created for the gingerbread character. We only managed to import the walk animation in the game. At first the bullet was supposed to be fired from the gun but since we switched to laser lines the bullet with animation is used as decoration on the gun now. (Kwok - Character design + animation, bullet; Coen - Fork, Gingerbread man model; Ayyoeb - gun; Kees - robot)

8.2 Audio

We spent a lot of time on the audio, because we believe that good audio will improve the overall quality of the game and it also creates a more satisfying game experience. We have sounds for shooting the gun (kill laser and gravity-switch), jumping, stabbing and hitting someone. The sounds were made

as 3D sounds to get an orientation of 'what happened where' in the level. There are also sounds that the players only hear themselves, like getting hit or stabbed, dying and respawning. Finally, there are also sounds 'of the game and level', such as: the moving of the robots, the menu click sound, and the sound that can be heard (eating a cookie and a rolling coins, depending on which game mode was played) when a game is finished.

Some of these sounds are generated ourselves in VSTi's (virtual Studio Technology instrument), while others were downloaded with a free license from the internet and afterwards modified with mixing techniques such as reeverb (to create a spatial feeling), parametric equalizing and filtering (cutting and modifying frequency) etc.. A lot of time needed to be invested in the sounds, because they repeatedly had to be changed or replaced. There are also a lot of sounds that we eventually decided not to use, due to changes (in weapons, movements etc) in our game.

Figure 8: VSTi's and mixing/master tools used

The most time, however, was spent on the production of the soundtracks. Different patterns had to be made with the rythym, bass, leads, and other generated sounds. All these individual sounds and loops had to be mixed, whereafter the whole track could be mastered. The first soundtrack, the menu-music, will be played while navigating through the menu. It is mysterious, not too heavy and well suited in the science-fiction/futuristic theme of the game. The second soundtrack is the in-game music. It is motivating, adrenaline-boosting music, has up-tempo parts, variations in rythyms/melodies and fits well in the theme. However, if the gamer doesn't like the soundtracks, they could be disabled in the settings menu. The sound-effects can't be disabled. (Ayyoeb - all; Roberto - programming)

8.3 Interface

There is also a score screen designed for in the game. We decided that we want to go with a transparent screen which appears when the player presses the 'Tab'-key. A few concept screens with different opacities where made in Photoshop and after that imported in Unity. Finally the design was passed to the programmers who made it able to show the actual live-score on the score screen. (Ayyoeb - design, Kees - implementation, Roberto - programming)

Figure 9: Ingame Scorescreen

8.4 Textures

We have made a lot of textures for the game. There are four different textures for each of the player models. Since it is a multiplayer game, it is important to be able to easily distinguish between players. At first we were using a tileable texture for all the blocks that make up the level. It looked very repetitive and tedious. So we made a number of different textures that are tileable and seamless, but are not the same. The blocks randomly select one of these textures when they are instantiated. This made the level look much nicer. But we eventually decided that there was not enough variety yet. So we made a number of extra textures that were applied to different sections of the level. These are the red lava texture, the pink crystal texture and the green grassy texture. These were all hand-drawn. (Kees - hand-drawn block textures, animated robot textures and gingerbreadman textures; Coen - level block textures, death boundary textures and skybox texture; Kwok - Lasermode character textures)

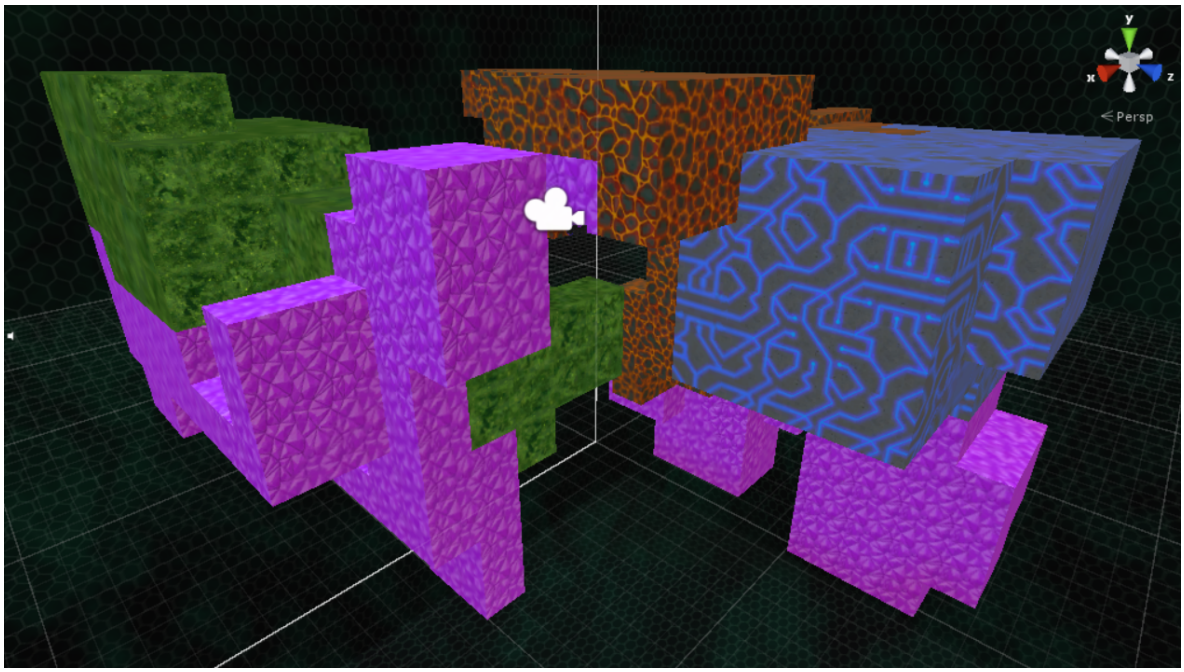


Figure 10: textures in level

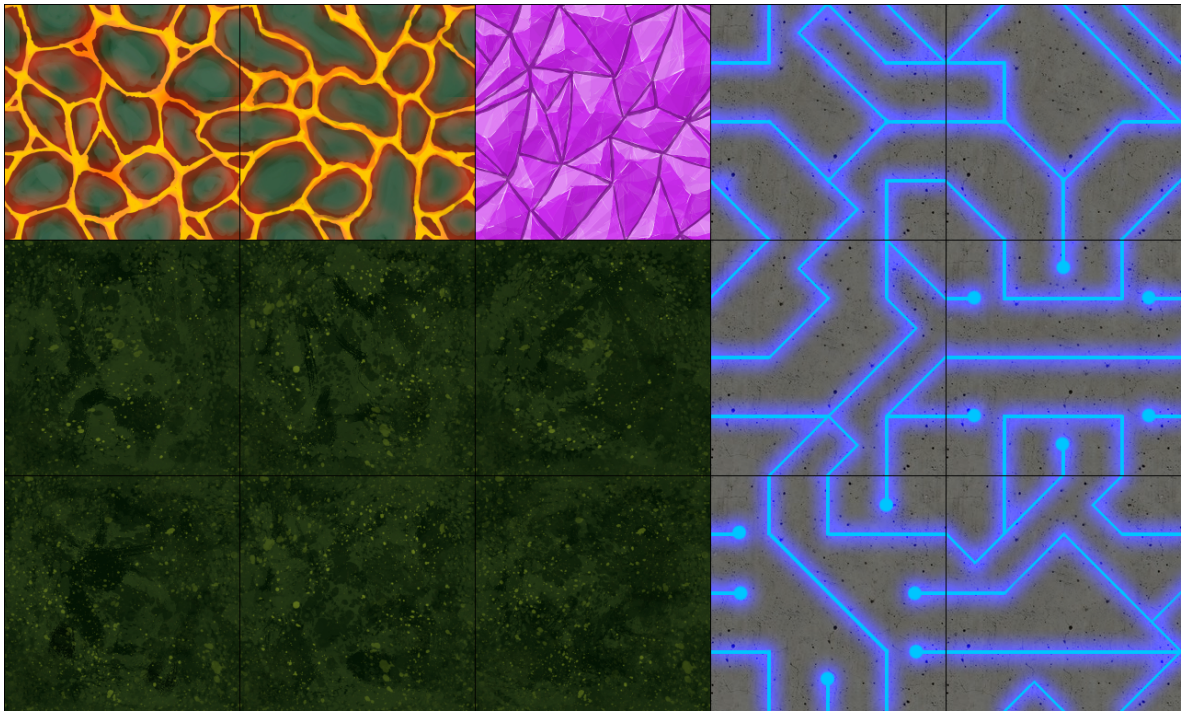


Figure 11: Overview textures for the blocks

9 Process

Because it was mandatory, we made use of the issue tracker inside Github. This is however not a success in our opinion. The amount of proceedings required to view or edit was too much compared to the benefit. We can image though that the issue tracker would live up to its potential in a bigger group. Both whatsapp and email provided a more efficient way of communicating bugs and issues in our situation.

We also made our own custom shaders to mix textures in Unity with different sizes to create a random effect. This was used for a while, but eventually we ended up handpainting the textures, rendering the custom shader useless.

We have created a lot of textures that we ended up not using. In the next picture you can see a sample of the scrapped textures.

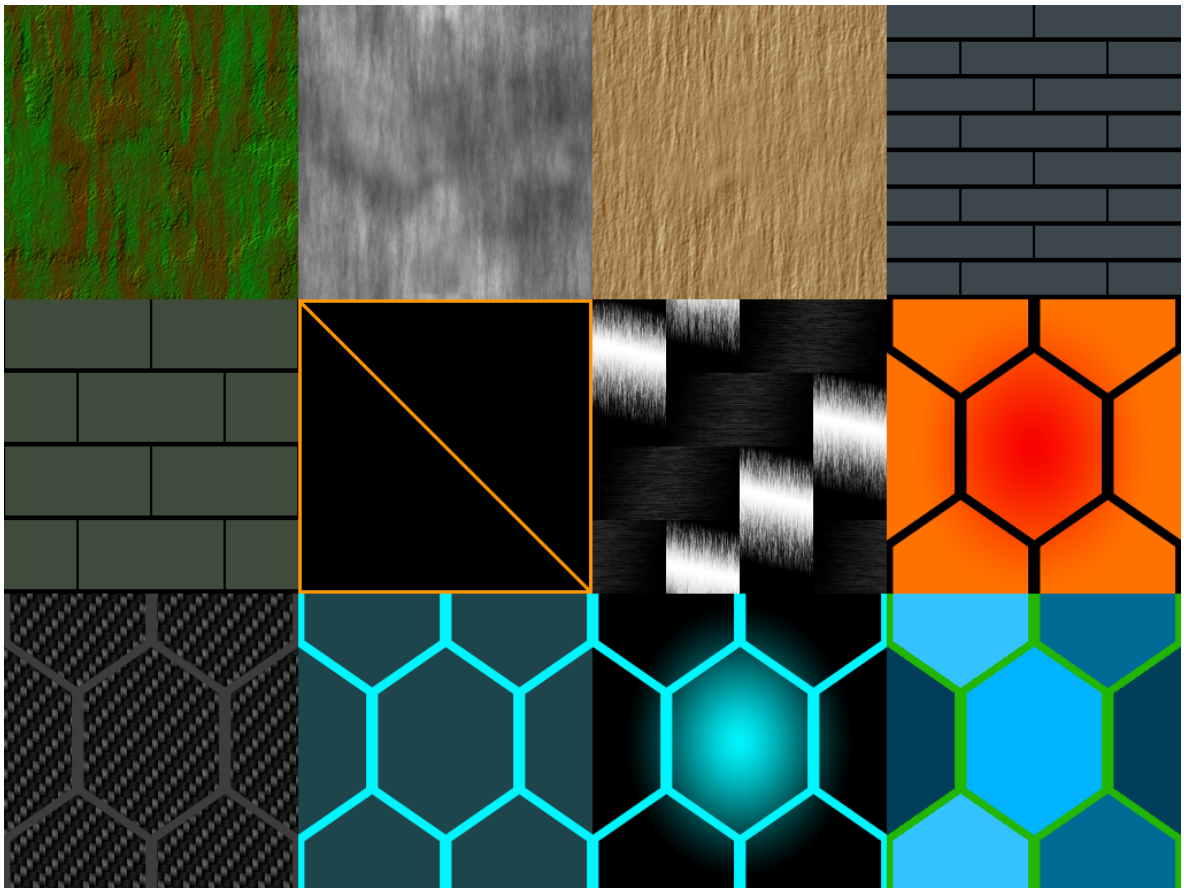


Figure 12: Scrapped textures

Before we actually started making the blue line tile textures, we created a simple concept. This concept was discussed in the group and that's when we refined the concept.

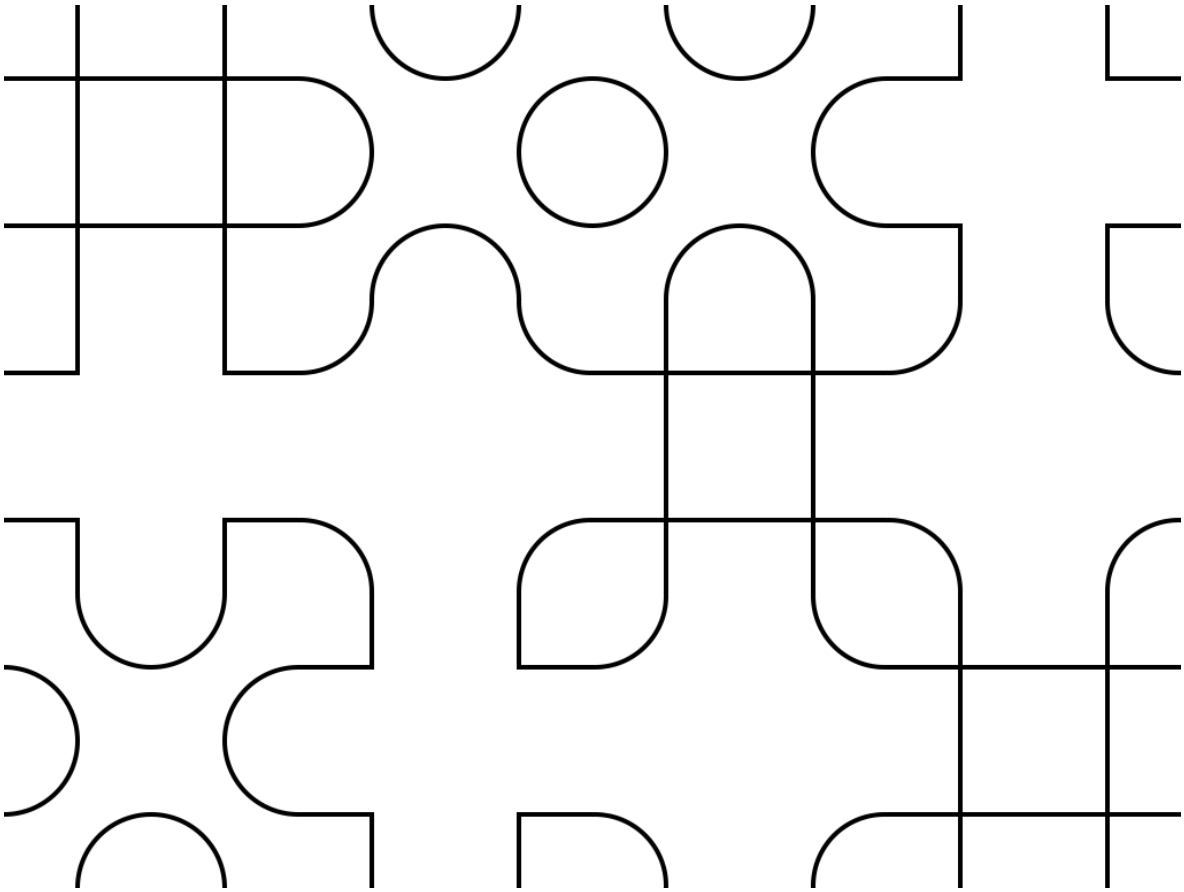


Figure 13: Tile textures Concept

10 Conclusion

And there you have it! We proudly present our game to you. In the last months we dreamed, iterated, tried, got stuck, helped each other, played, and just had fun. If we would restart the project we would have a lot more experience to start with, which is expected. But we would not change up the workflow a lot. It served us very well. We are proud of the final result and enjoy poking each other into a pulp in our own multiplayer videogame.