**LaserStorm**

A First-Person-Shooter

Project-customer: DI Günther Senfter

Project-Leader: Elias Gendu

Project-Team: Florian Griesebner, Jonas Hörtnagl, Tobias Muigg

**The Aims:**

* We fulfilled all the must haves.
* 3D-graphics
* Connection in a local network
* Movement of players
* Ability to shoot
* Ability to hit foe with a projectile
* Dying after getting hit
* Lasers as shots

These points have been implemented.

* We had defined some points that could be included.

These points have been implemented:

* Connection over the internet
* Lobby System
* Out-of-game UI

These points have not been implemented

* Different guns with different behaviours/playstyles
* A save function for statistics
* A team-mode
* A chat

**The requirements**

These points must be included that the game can be played:

* A computer with Windows as OS
* A mouse
* A keyboard
* Internet connection

**How does the game work?**

It is a First-Person-Shooter for multiple players.

Every player has a gun.

With a left mouse click the player can shoot.

The projectile is visualized by a laser.

The player can walk. Therefore, the keys W (forwards), A (left), S (backwards) and D (right) or the arrow keys are used.

If the player would like to run he or she just has to hold down the shift-key additionally.

The player can also jump. The space-key is used for that. The player can jump and jump one more time while being in the air.

Thereafter, the player will fall to the ground again.

To turn around or look up or down the mouse can just be put in that direction.

If the ESC-key is pushed a menu appears the player can click on the back-button get back into the lobby. If the ESC-button is pushed one more time the menu will disappear again and the player can go on playing.

**The implementation:**

We used 3 Programs for our implementation:

* Unity was used to implement the game
* Blender was used to create the player-avatar and the animations
* Monodevelop or VisualStudio to create the C#-scripts

The C#-scripts are attached to the game objects to create scene-changes, movements

3 different Scenes were used to create our application:

* Lobby-Scene:

This Scene has the purpose to

* host a Server
* join a game
* choose a name
* choose a colour
* Game-Scene

This Scene is the main Scene.

Here is the game played.

****

* Score-Scene

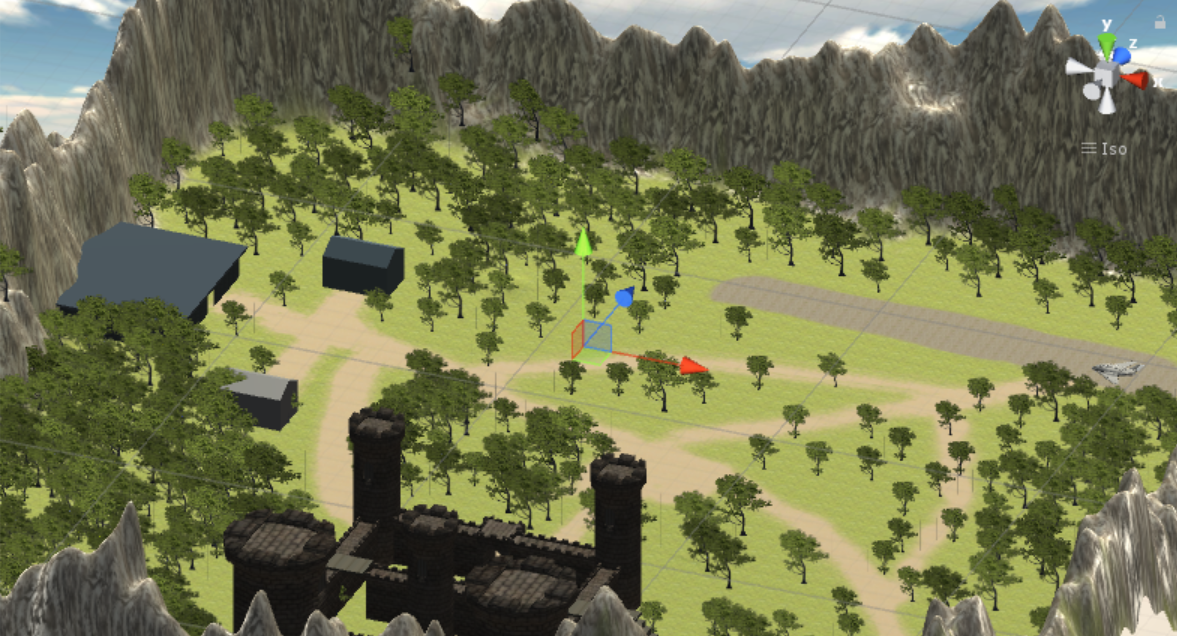
This Scene shows the score after the game has been exited.

These statistics are shown:

* Names of the player
* Amount of the hits that the player made
* Amount of the hits that the player got

**Creation of the area and the map:**

At the first attempt, the team agreed on a map which plays in the forest. This would give the game a dynamic design, as the plan was to implement effects as a wind for example.

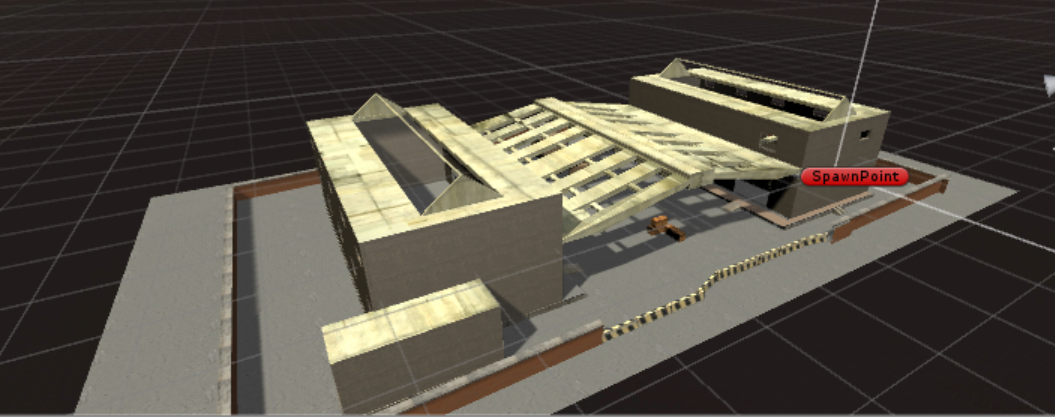


The concept was finished and the screenshot indicates basically how it should have looked like. It was planned to create two spawn-area on both opposite sides of the map, this would have been where the players were revived and start again after getting shot. Also a small castle and an old airport should have been there.

After analysing the first design of this forest map, the team observed that the layout of the too large. Because in case there are only two players playing, the most time would be spent on running around without seeing the other player.

This recognition leaded the team to a decision. The forest map would be deleted and replaced by a smaller map with another layout. A problem of this decision was the delay of the time plan, which was made before the project started.

The new map was implemented in a complete other style. It should show a storehouse with some objects spread over the map, which allows the player to hide while a shooting. The following screenshot shows how the new map looks like.



In comparison with the forest map this one has the advantage of being smaller. So the player have more contacts and the play feeling is faster.

The maps were designed with a special 3- way-principle. This is also used in the professional gaming business. For example in Call of Duty or Medal of Honor

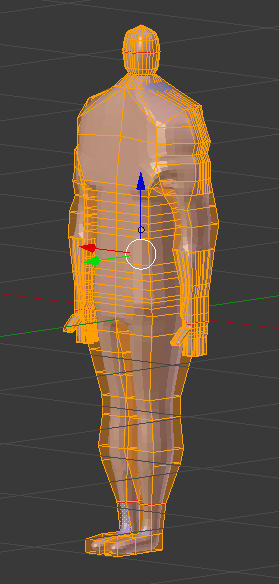


It describes the layout of the map. The spawn-areas should be connected with 3 ways. So the players have the best play feeling. It does not take too long to meet the other player, so the player are not bored and it is not fast, so the player does not get confused. The 3 ways do not need to be complete separate from each other. In our case the middle of the map is relatively open and the 3 ways connect their.

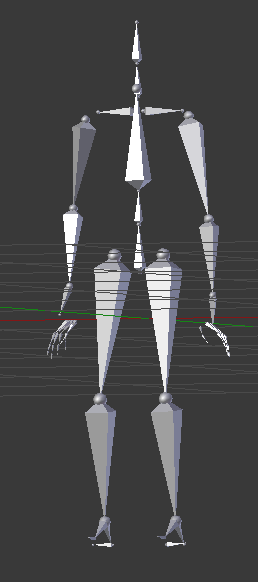
**The creation of the player-avatar:**

Created was the player-avatar with Blender.

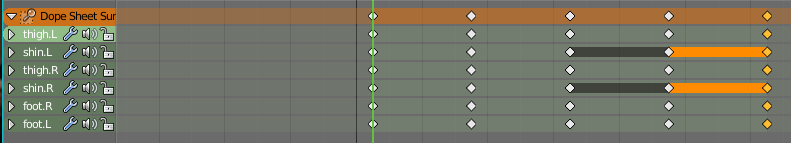
The player-avatar was modelled.



For the animations an armature was created and the created mesh was combined with armature.



Thereafter the animations had been made. In the DopeSheet were while using KeyFrames 3 animations created:



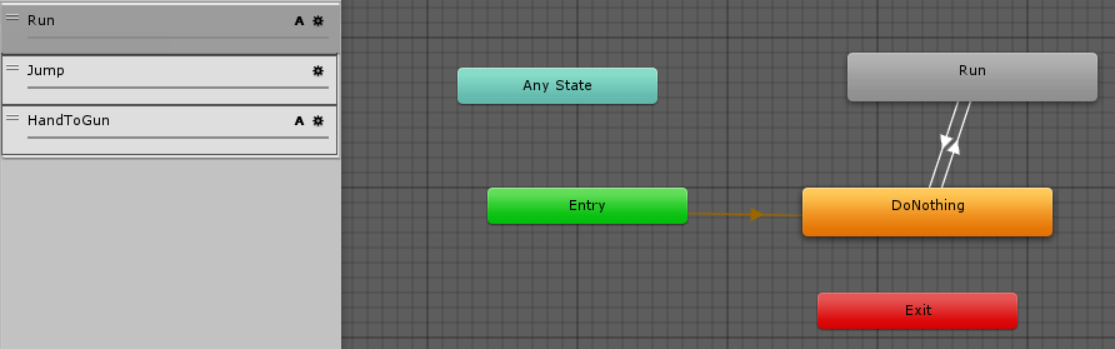
* Jump
* PutHandToGun
* Walk

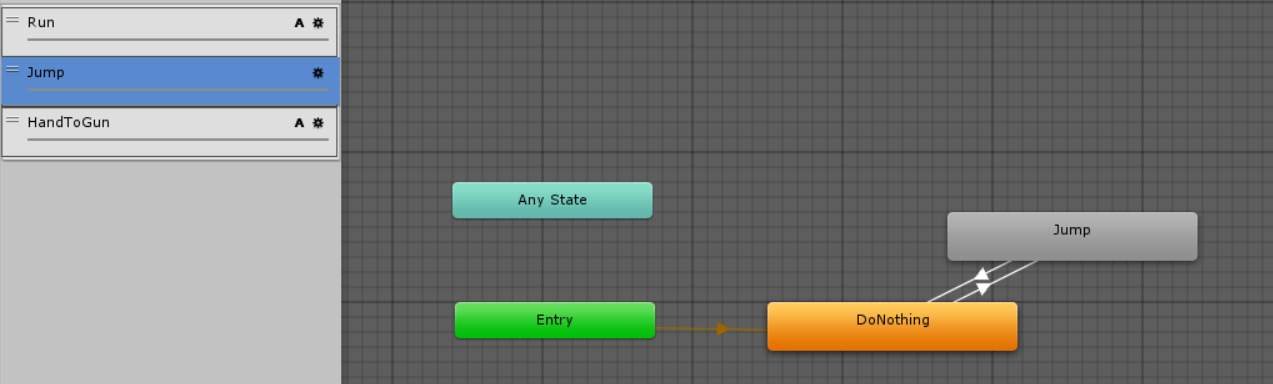
The Blender-project is exported as a .FBX-file and in Unity put into the Assets-folder.

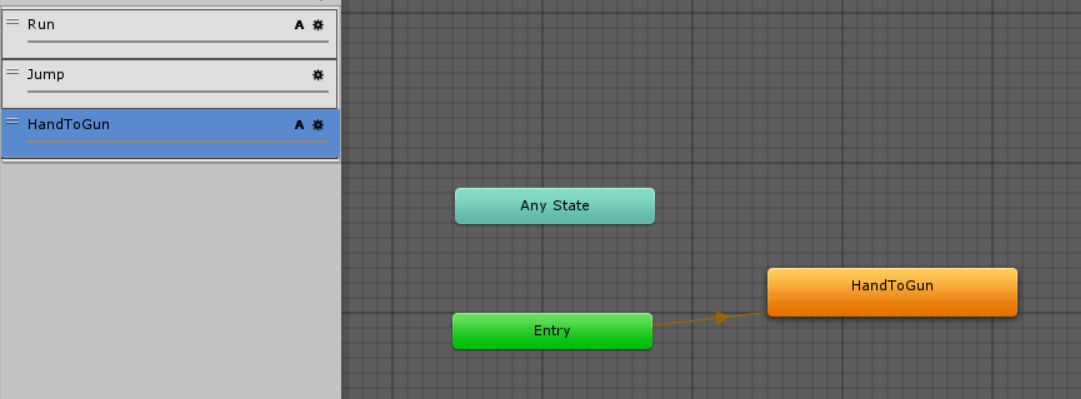
**Animator in Unity:**

With the animator in Unity there is a kind of finite-state-Machine created.

Every state can get an animation-clip.







**Server:**

Unity does a lot of the low-level tasks for the programmer.

The network manager has to be set up. It handles all the connections whether they are to other users (local network) or to the unity relay servers.

It has to be decided what properties will be synchronized over the network.

**Things that consumed more time than expected / Problems occurred:**

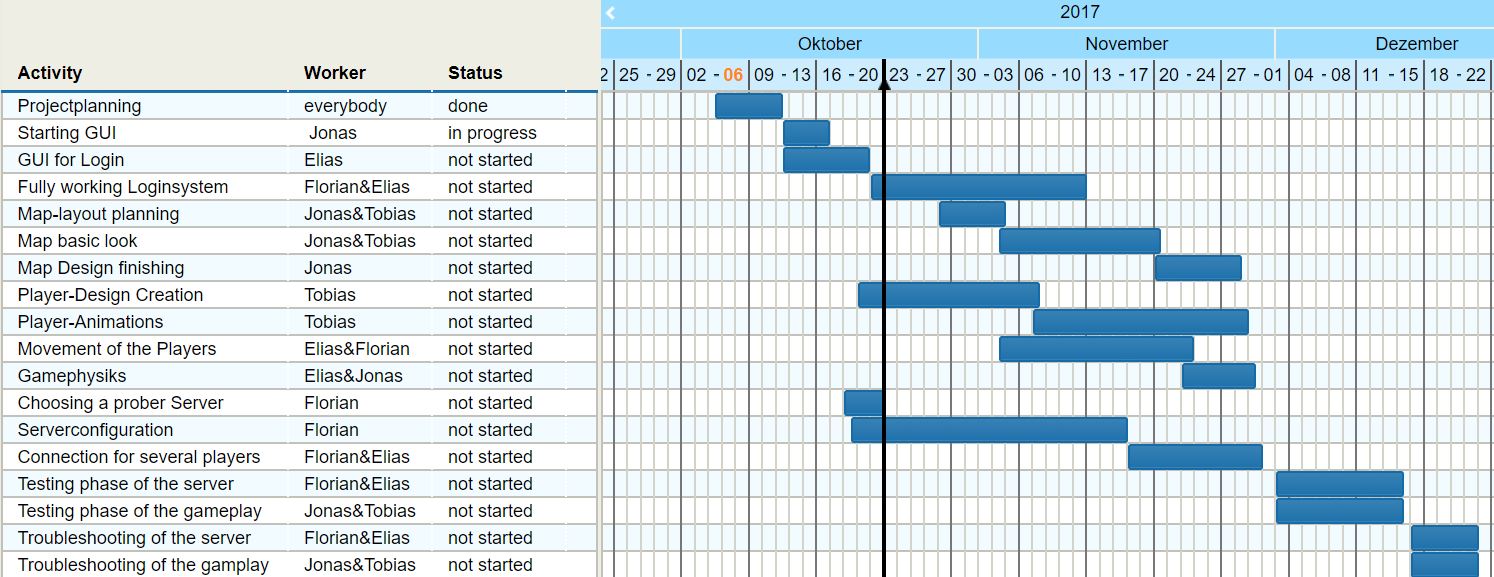
* The name tags were more work to implement than expected. The tags must turn always towards the player that is looking at it.
* Animations over the server
* Shooting over the server
* Player creation (armature always interfered wrong)
* Collisions with other objects

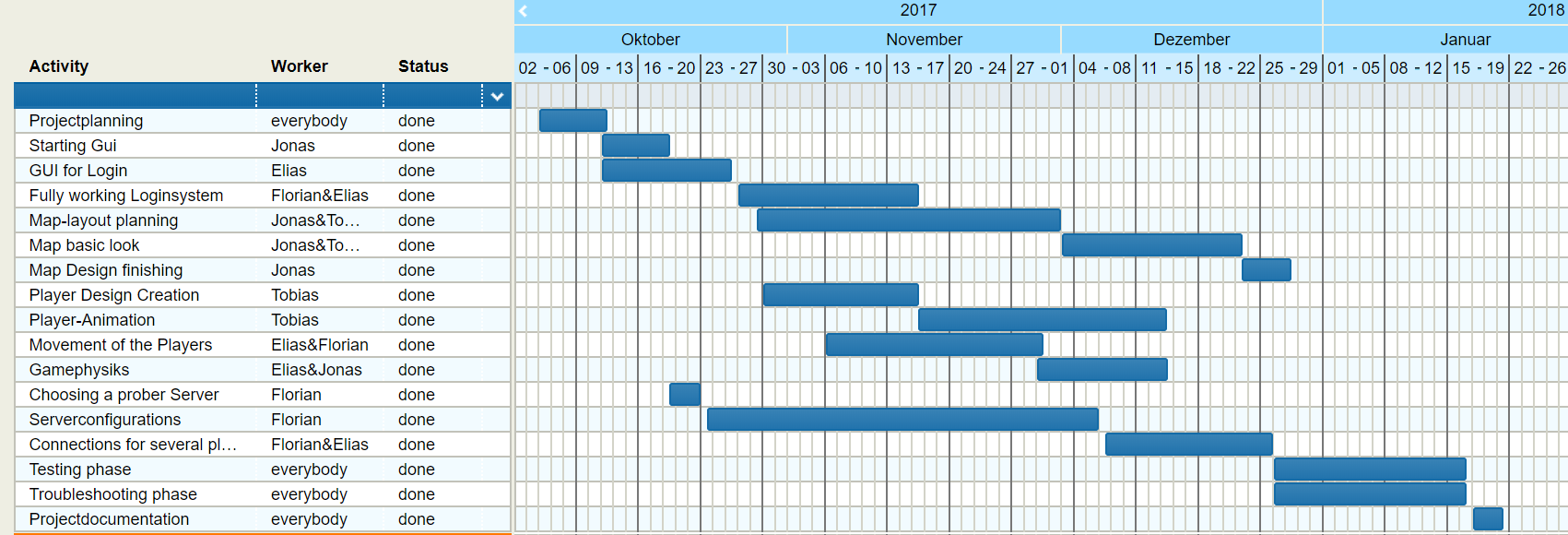
**Things that were implemented far more easily than expected:**

* Game online over a server

**Time planning**

The following GANTT-diagram shows the planning of our time management.



The following Gantt-diagram shows the time plan how it actually happened.