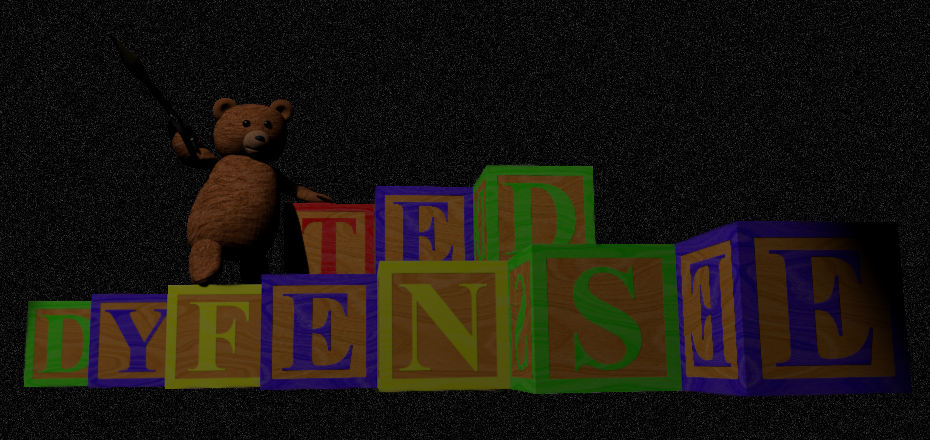
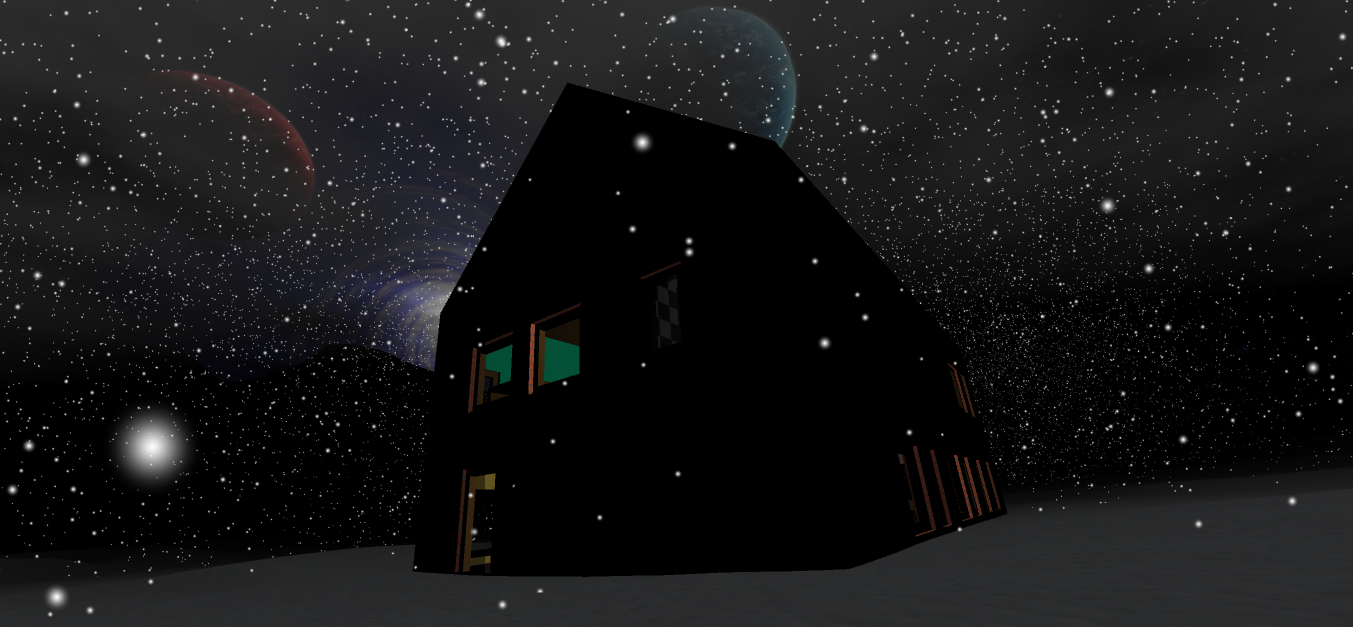
****

TedDyfense





Development team:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Stud. Nr. | Study | Email adress |
| Abdullah Tezcan | 1353306 | Bouwkunde | a.tezcan@student.tudelft.nl |
| Jeroen Methorst | 4246330 | Nanobiologie |  |
| Arjan van Ramshorst | 4023005 | Werktuigbouwkunde | A.B.vanRamshorst@student.tudelft.nl |
| Esmeralda Tomasöa | 4004329 | Technische Natuurkunde | e.c.tomasoa@student.tudelft.nl |
| Damien Crielaard | 4260120 | Technische Natuurkunde | D.R.Crielaard@student.tudelft.nl |
| Pieter Kools | 4134451 | Werktuigbouwkunde | P.J.S.Kools@student.tudelft.nl |

**Introduction**

This document will introduce the reader with TedDyfense, a game we have been developing the past 10 to 12 weeks in order to complete our minor project. The document will focus on the technically challenging components, the code quality and its management, the art and design process and finally the development process we used to achieve our final result.

**Target Audience**

Usually the main audience would consist primarily out of male players, but due to the trend of increasing female gamers in the past years we made our game design more appealing to a more broad audience.

**Platform and Controls**

The game has mainly been tested on windows platforms, and any debugging was focused on achieving a working Windows build. We have also tried a few Mac builds which worked without problems.

The controls are simple and resemble the average shooter, with a few additions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Key** | **Action** | **Key** |
| Movement | W,A,S,D | Place block | F + Left Mouse Button |
| Jumping | Space | Remove block | F + Right Mouse Button |
| Running | Left Shift | Pause game | P |
| Fire weapon | Left Mouse Button | Reload weapon | R |
| Aim through sights/scope | Right Mouse Button | Switch between 1st/3rd person | F5 |
| Change weapon/blocks | Middle Mouse Button (Scroll) | Enable Torch | T |

**Story and setting**

The story is about a young child, "Little Johnny", having a nightmare and the only thing standing between him and his nightmares is the player, his Teddy bear.

As Little Johnny's teddy bear, you are tasked with defending Little Johnny from his own nightmares. To accomplish this, various types of heavy weaponry (Colts, AKs, Rocket Launchers, Shotguns) can be found throughout the house for you to use. But don't stray too far from Little Johnny or his nightmares will get the best of him!

**Technical Components**

Weapon and ammo systems – Damien

The weapon system was designed around the idea of versatility. The stats for the different weapon types can be set individually. These stats include things like base damage, clipsize, rate of fire and type of ammo. For each type of ammo an ammo pool is made. This way multiple weapons can share the same ammo, although this is not used in the final version.

Special weapon behavior – Damien

The weapons can have a custom script that defines their specific behavior but all weapon-scripts inherit from the weapon superclass. In this class behavior that is shared amongst all different weapons is defined. There are several types of weapon behavior scripts in the game. The first one is the SimpleGun for the rifle and pistol, which is a simple damage dealing at ray cast hit. Secondly, the SimpleShotGun script is implemented on the shotgun, which uses several ray casts. The RPG script is used for the sole projectile based weapon with an area of effect type damage. Last but definitely not least is the LandMine, which can be placed on the ground and then explodes on contact with an enemy or player.

Loot system – Damien

The pickups that the enemies drop on death can be set per enemy type. The drops are set using a master probability of dropping loot and an individual probability for each drop. This way certain drops are rarer than others and are more likely to drop from certain enemies than from others.

Health and damage system – Damien

The implemented health scripts control the health of Little Johnny, the player and the enemies; but all inherit from the same health class. Hitting certain body parts does more damage than others e.g. headshots are often a one hit kill. The player regenerates a small portion of his health whereas Little Johnny regens to full, albeit at a much smaller rate. The enemies do not regenerate hit points. Different enemy types have different health characteristics.

Block Placement – Damien

Blocks are placed in the world by using a ray cast and placing it at the closest possible position to the hit point in the direction of the surface normal. To indicate this to the place a green or red transparent block is shown at that position to indicate valid or invalid positions respectively. The blocks that the player has are finite but one is added each time the player kill an enemy.

Toon Shader- Arjan

Because of the cartoon like backstory to our game it seemed logical to implement an unrealistic looking shader to emphasize the surrealism of the environment. This was done by adding a threshold value to the shader. If the intensity of the diffuse lighting exceeded this value the shader will interpret the surface as lit, otherwise it will be interpreted as shadow.

Log in Screen – Arjan, Damien

The first ingame screen is the log in screen. The player has the choice to submit a new username/password combination and/or log in. When the Server is off-line or the player wishes to play without registering the Skip button can be clicked to continue as an Anonymous player.

HighScore Logging – Arjan

A Highscore system was implemented into the game. Logging statistics such as accuracy, amount of shots fired, amount of headshots and so on provides the player with a simple overview on how his game session turned out.

WorldBlockManagement script – Pieter

The WorldBlockManagement script contains the fundamentals for our in-game world. This world is divided in 1m3 blocks which can be:

* Placed by the player to move around and defend against enemies.
* Removed by the player.
* Removed by the enemies (Refer to the EnemyAI script part).

To build the world, it is either possible to generate a new level with the given dimensions, or load one from binary file. The script also contains a method that writes the current level to a binary file. As the enemy pathfinding only uses this script to get its data from, there is a blockifyworld method. This method places invisible blocks in places where colliders from the house model are.

pathFinding script – Pieter

The pathFinding script is used by the enemies to find their 3D path to:

* The flag (around user-placed blocks).
* The flag(breaking user-placed blocks).
* The player (around user-placed blocks).
* The player(breaking user-placed blocks).

The script will create an (invisible) overlay over the world with a gradient field, with the target having a value of 0. This way, at each point in the world by looking at the 8 blocks around you, you can simply get a direction and distance to the target. An advantage of using this method is that multiple (infinite) enemies can use the same overlay to find their path without performing the same computation for every individual.

EnemyAI script – Pieter

The EnemyAI script controls the behavior of all enemies. It chooses which of the 4 pathfindings will be used and why (For example: If it is less than 20 blocks away from the player or if it can see the player, it will pathfind to the player). It will also start breaking blocks in front or above the enemy when certain conditions are met. This script also contains everything that causes the enemies to move, jump and fall.

EnemyController script – Pieter

The EnemyController script contains the “Wave system” we use to spawn the enemies. It spawns randomly chosen (with some probabilities) enemies on the edge of the level and keeps a list of them. When all enemies are dead, it will spawn a new wave after a set-able delay.

EnemyRadar script – Pieter

The EnemyRadar script gets all enemy positions from the EnemyController and plots them on the UI radar sprite if they are within range. The radar also rotates, so that the space above the player on the radar represents the direction he is looking at.

gameCameraSelector script – Pieter

The gameCameraSelector script allows you to toggle between first and third person view. It redirects all methods to either the first or third camera script. It also draws the crosshair to the middle of the screen.

FirstPersonGameCamera script – Pieter

The FirstPersonGameCamera script contains all the functionalities to move around the world in first person and weapon-related things. These things are handled by the script:

* Camera movement.
* Recoil when firing.
* Weapon reloading animation.
* Weapon switching animation.
* Aim down sight.
* Aim down ACOG (scope) sight.
* Aligning the weapons with the camera (also while aiming down the sight and scope).
* Sprinting and sneaking (You cannot reload/switch/aim down sight while sprinting).
* Jumping (physical corrent).

ShooterGameCamera script – Pieter and Damien

The ShooterGameCamera script is the third person camera script which initially was our only game camera script. This script handles:

* Camera movement.
* Recoil when fireing.
* Aligning the weapons with the teddy model.
* Sprinting and sneaking.
* Jumping

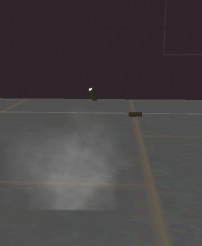
**Code Quality**

To maintain code quality all the programmers added comments to organize their code and stuck to the standards that were taught in the OOP course. When altering code the editor consulted the creator to minimize the chance of overriding or deleting vital parts.

**Art, Design, and Style**

Weapon effects – Damien

When a gun is fired the player receives feedback in the form of several different visual effects on the screen. At the end of the barrel a muzzle flash, consisting of a point-light and a fire effect, is shown. At the point of impact a spark effect, a puff of smoke and a small impact flame effect similar to the muzzle flash is shown. The guns also eject a bullet shell from the side of the gun. The shell is instantiated using a generic object pooling script to minimize performance cost.

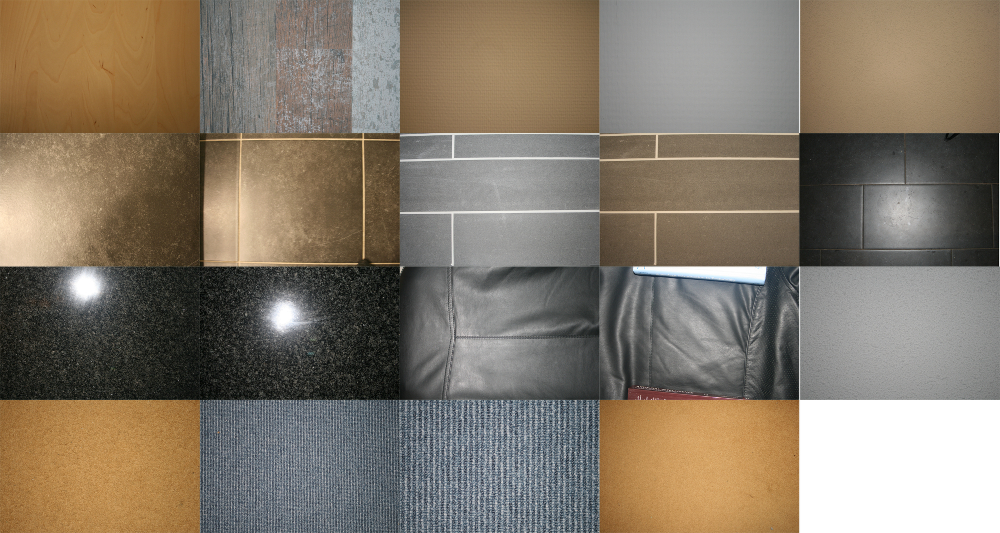
****

The screen also gets a red overlay that fades over time when the player has been hit recently.

Custom Sounds – Arjan

Some in game sounds are custom made. Explosions, gun firing, blockplacement, menu transitions and menu clicks are actions that implement these custom sounds. By recording voice sounds and using common household objects the raw data was recorded. Afterwards the raw data was processed using Audacity. Using Pitch-Correction, Reverb and custom equalizers the final result was achieved.

Textures - Jeroen

Most of the textures in the game have been created from photographs made by myself. These photos were digitally edited to reduce differences in brightness and allowing them to be tiled seamlessly.  
   
*(Small example of the different photos used in texture creation)*

A small amount of base textures were taken from CGTextures. These textures were used in the creation of larger textures. (for example the planets in the skybox, these were made with small parts of 5 different textures). The sources of these 3rd party textures have all been documented and can be found in the GitHub directory (\Graphics\Textures\\_Base\_Textures\3rd Party\Sources.txt).

These textures have been used everywhere in the game. Walls, floors, furniture, weapons, the player character and also the enemies.

Skybox - Jeroen

The game's skybox was created from a large texture built up from several smaller textures. The planets were created from base textures which have been projected on top of a sphere, while the galaxy was created using a rotation script in GIMP. The stars were generated from noise.

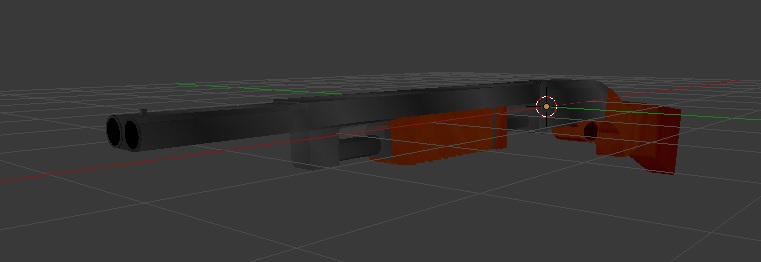
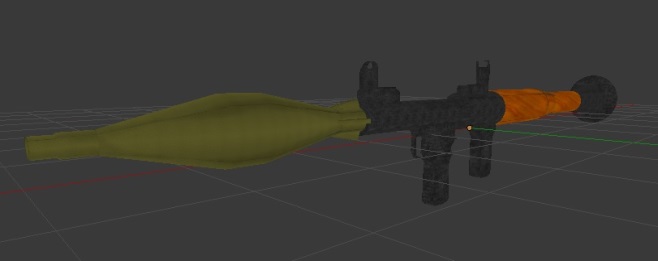


To create the 6 skybox textures, the skybox textures were imported into blender and projected on top of a large dome covering a cube. Environment mapping was used to map the dome onto the cube.



Weapon Models and Textures - Jeroen

The weapon models were created using Blender by tracing over photographs of the actual weapons from different angles, this process took some time but resulted in accurate models of the weapons.



The textures of the weapon models were created by UV-mapping the individual parts of the weapons in Blender, exporting the UV-mapping to a .png file, and then adding textures to the exported UV-map.

*(Example of the Shotgun Texture)*

House – Abdullah

The house and the furniture are created with the program SketchUp. The house is made in a grid of 1 by 1 units to work with the blockify function. The enemy’s can jump through the window and find their path to little Johnny in the house.

The house has two levels. The stairs connects the entry level with the first floor where you can find two bedrooms and the bathroom. The master bedroom has stairs connected to the entry level.

Jeroen placed the textures of the furniture, walls and the floors. Damien placed the model in to the Unity file and added the Mesh Colliders.

User Interface – Abdullah

The UI elements are created with Adobe Illustrator and exported as \*.png file. In Unity they are converted to a Sprite, which they can be added in to the scene.

 Button

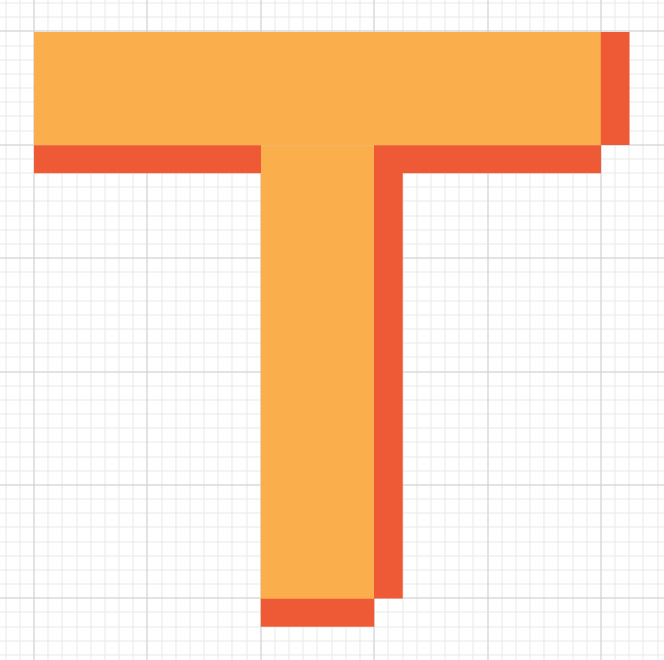
 Hover

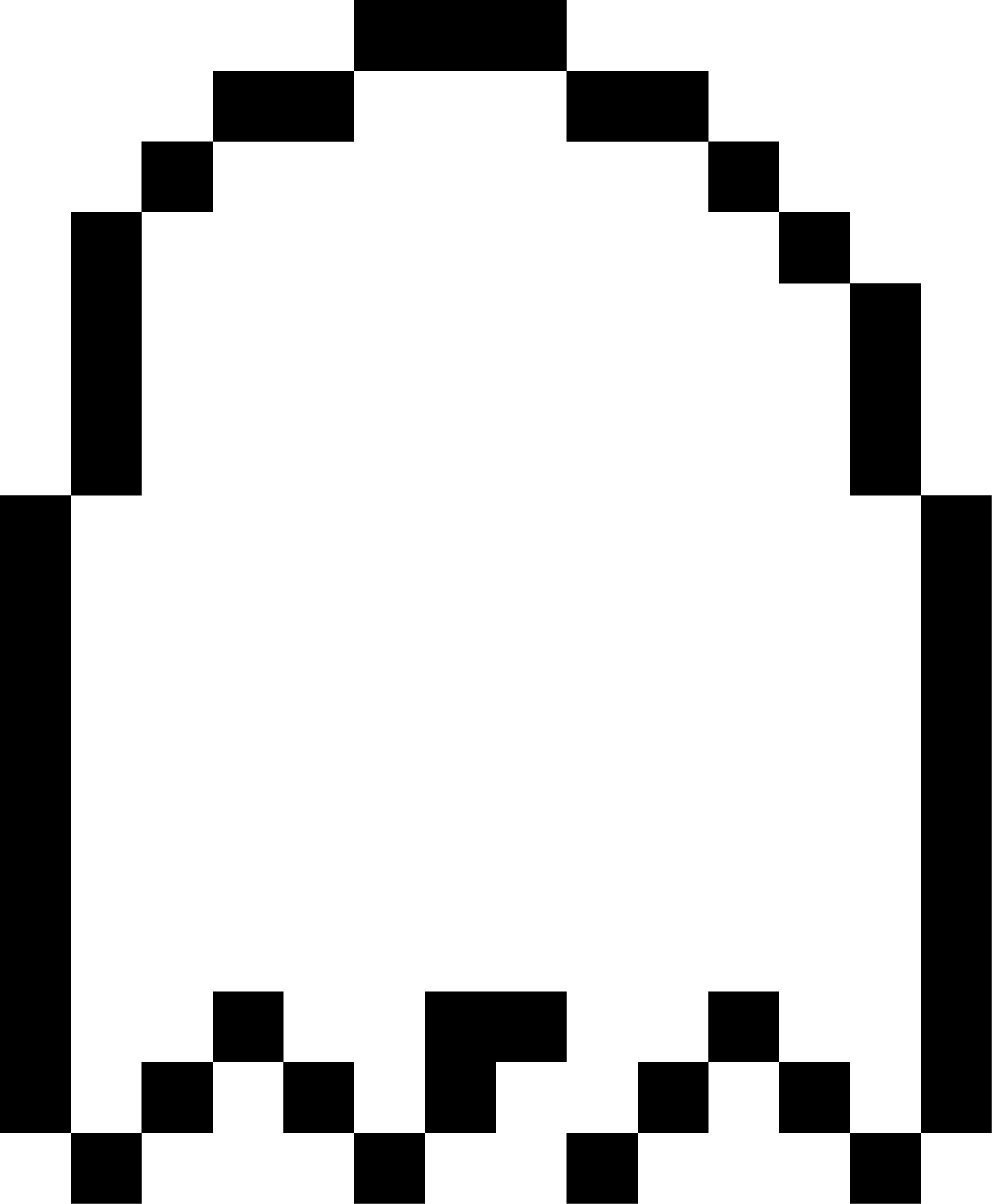
 Click

 Disabled

The buttons change when they are in interaction with the mouse. They have 4 stages as shown above.



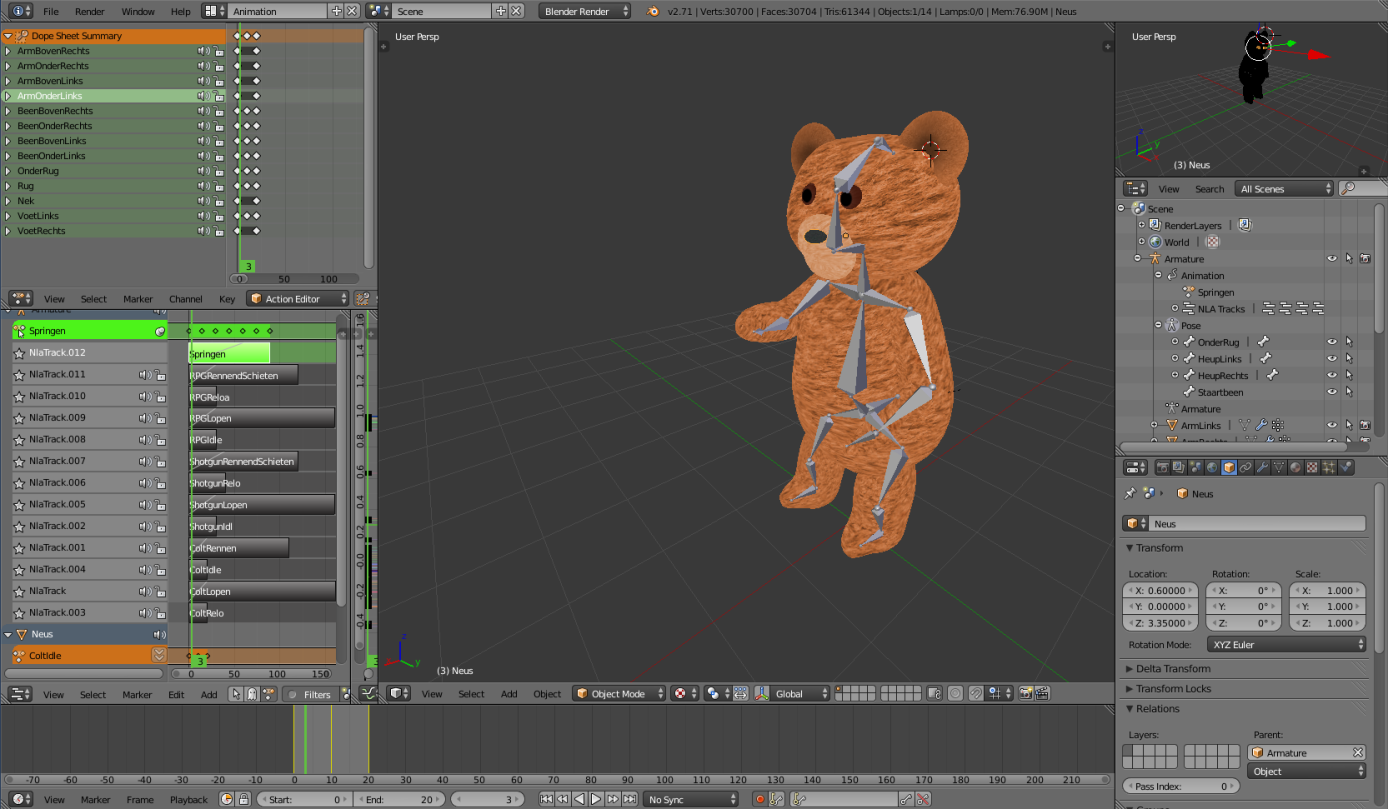


The characters are created in a 5 by 5 grid as shown left. With the shadows the characters gets an 8-bit look.

The enemy counter at top right of the screen is also made in 8-bit style.

Teddy, main character - Esmeralda

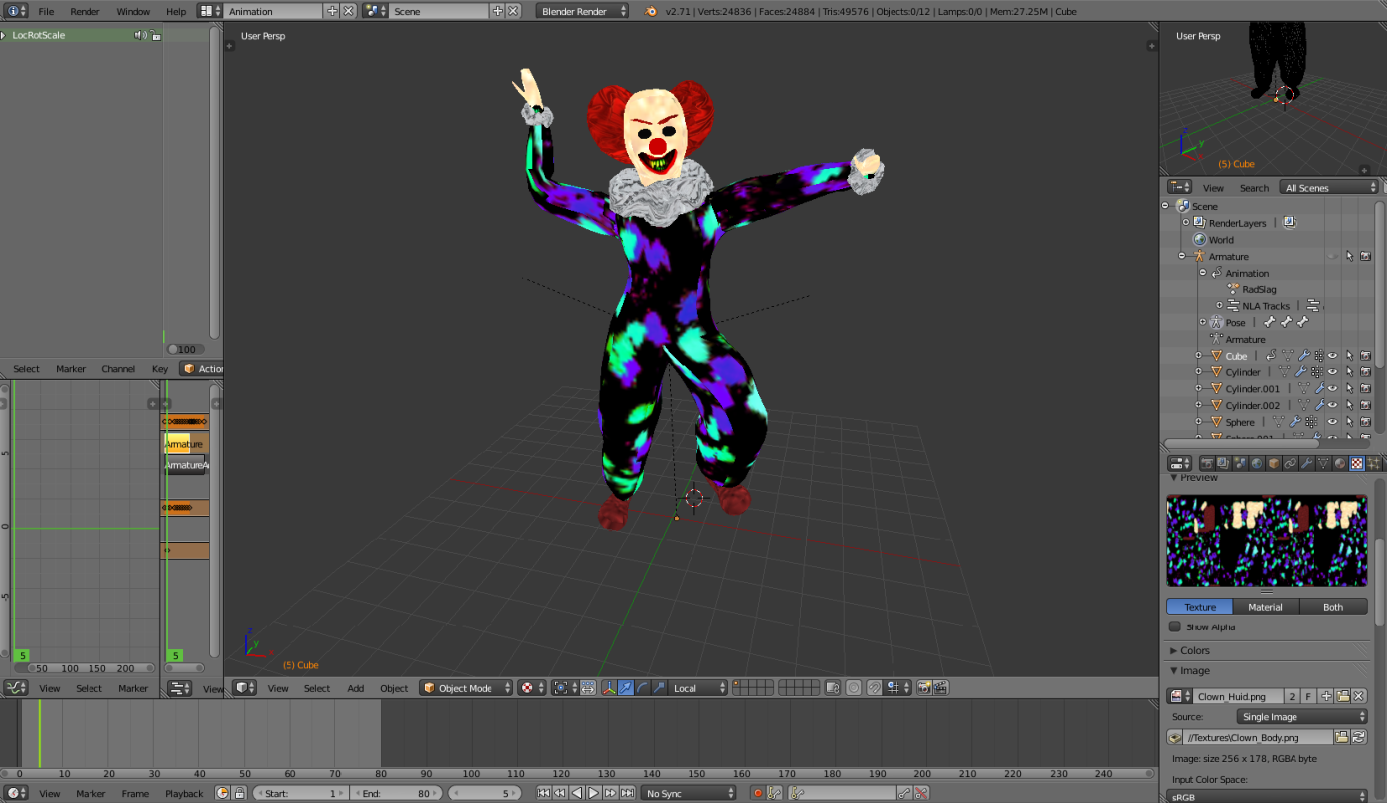
Made in Blender by sculpting multiple spheres. The model is animated, using bones in blender. The produced animations are: Walking, Running, Dying, Jumping, Running/Shooting while holding RPG, Reload RPG, Walking while holding RPG, Running/Shooting while holding Shotgun, Reload Shotgun, Walking while holding Shotgun, Idle holding Shotgun, Running while holding Colt weapon, Idle holding Colt, Walking while holding Colt, Colt reload.



The textures are by Jeroen

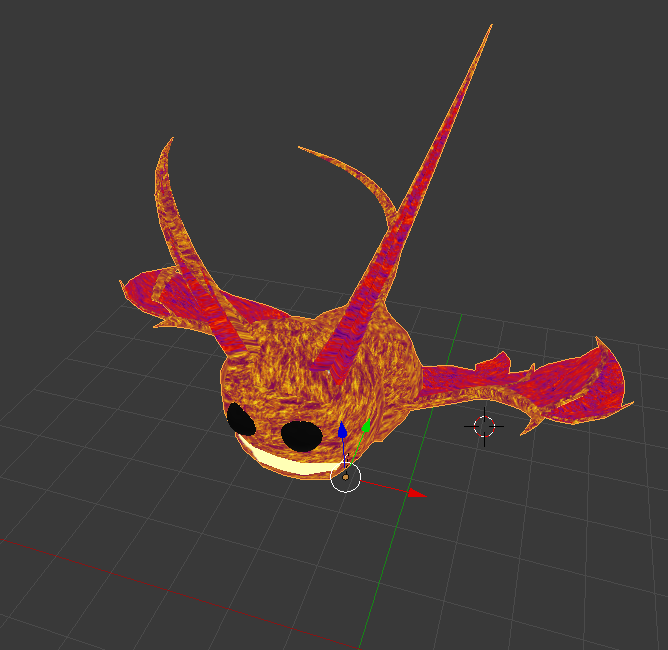
Clown Enemy – Esmeralda

Made in Blender by editing the vertices of a start object (cube). Using this method a simple character was created. In Sculpt mode the function inflate was used to give the illusion of baggy clothes. Starting from cylinders the collar of the Clown was made. A part of the Clown his attire was given color by UV-mapping the texture on the model. This model is also animated, using bones in blender. The produced animations are: Radslag and Karatetrap.



Bat Enemy – Esmeralda

Made in Blender by editing the vertices of a start object (cube). This model is textured by editing base textures from cgtextures.com and using UV-mapping to put these edited textures on the model. The model is animated using the bones feature in blender. A single animation is made for this model: Fly.



Other enemies – Damien

The BlockBot and Snowman enemy are made using simple primitive. The Evil Teddy is the main character model recolored.

**Development Process**

Due to the complex nature of our weapon system and AI it was hard to deliver working code on a weekly basis. We tracked our progress using the issues on GitHub and made sure to meet on all scheduled occasions.

**Conclusion**

The final result is the game we intended to build. All the features we wanted for our game design were implemented, some with ease, some with bugs that never seemed to go away. It turned out that when it comes to programming, major issues should never be postponed in favor of fixing minor issues. People specializing in certain aspects of the project is efficient of terms were not everyone has to undergo a learning curve.