Overview

I wanted to recreate the classic „Moorhuhn“ game, released in 1999, where the player has 90 seconds to shoot targets and receive points, which are displayed in the highscore list.

Hitting enemies in the frontline is rewarded with 25 points, enemies that are far with 35 points. It is possible to hit a target multiple times for additional points.

Controls

Classic FPS controls, move around with WASD, space to jump, move the crosshair with mouse.
Left mouse button to shoot, right click to reload.

Reload is only possible if at least on bullet was shot, while reloading, shooting is permitted.

Fullfilled requirements and features

User-controllable character

Standard FPS character, WASD movement + space to jump, mouse controlled crosshair: left click to shoot, right click to reload; Uses a hitscan at the current mouse cursor poisition to detect hit

AI-controlled pawns/characters

- Moorhühner are spawned from EnemySpawner, controlled by AI-Controller;
- I have understood that behavior trees are used for changing AI behavior, especially in combination with connected animations -> since the opponents I use have no change in their behavior nor do change their animation, I decided instead to compensate the points by implementing different movement patterns in the AI controller;

Map / level

- menue system: main menue connected to highscores and level
- main level: floating rock in a 3D environment; character is spawned within a not passable (**BSP Brushes**) area where movement is allowed;
  Usage of **Sky Sphere** and **Light** for global illumination and mood; extra **directional lights** setup to illuminate player area and map;
  Contains a **FPS HUD** which displays remaining time, ammunition and scored points realised with UMG; Draw custom crosshair including hit marker;
  Also implemented a screenshake after each shot and a restriction for spamming

Custom models and materials

- use of different materials for background environment; player character weapon and enemies;
- because animation for custom Moorhuhn didn't work at first, walking animation for starter package Mannequin was used; after I get another Moorhuhn **model and material**, I used Mixamo to create a **animation** which was importet

Used external content

- Particle system for muzzle flash from Sci-fi Gun Pack by Quaternius (https://www.patreon.com/posts/free-sci-fi-gun-18871201)
- Particle system for explosions at enemy hit from UE4 infiltrator Demo (https://www.unrealengine.com/marketplace/en-US/product/infiltrator-demo)
- Environment such as mountains, plateaus, trees and billboards from UE4 infiltrator demo
- Moorhuhn created by Marlene (https://twitter.com/Merlar98)
- Crazy chicken created by Jorge Rodrigues (https://www.artstation.com/artwork/4bdZz8)
- sounds and music mostly from original moorhuhn game (https://store.steampowered.com/app/340120/Moorhuhn_Crazy_Chicken)
- additional sounds from weloveindies.com using student licence
- Moorhuhn remastered menue background image

Realization of logic through blueprints

- BP_GameMode for gameloop
- BP_FPS_Character implements game mechanics such as movement and shooting / reloading
- BP_Camera Shake to add firing feedback
- BP_Moorhuhn reacts to hits, plays animations and sounds; destroy itself
- BP_Moorhuhn_back (not used anymore) was the Enemy implementation with starter content mesh and animation
- BP_AI_Controller move the enemies depending on mode through the level; split different behavior in seperate functions
- BP_Enemy Spawner periodically creates new enemies
- BP_Save Game struct to hold gamestate

C++ integration

Helper class for sorting as key-value pairs to show in highscores

Extra Points

Music and Sound Effects

- different ambient sounds for menue and level
- sound effects in level for shooting and reloading, as well as enemy hit and countdown
- menue sounds for button clicks

Particle Systems

modified (but not fully implemented) a particle system fort he muzzle flash as well as for exposions on hit. Additionally, I implemented a particle system for enemy disintegration on despawn a few times by different guides, but they didn't work out.

Saving and loading game data

Because ingame save slots did not fit well into the game, the GameSate saving system was used to store and load highscores

References

- assets stores http://shootertutorial.com/assets-stores-links/
- animation creator from static meshes https://www.mixamo.com/#/
- retarget Mixamo animation to UE4
  https://www.youtube.com/watch?v=UoGdTouVeRs&ab_channel=CARD00R
- Godot Moorhuhn clone implementation
  https://talesfromimdahl.de/2019/04/04/how-to-part-2/
- Disintegration effect
  https://www.youtube.com/watch?v=gldIJGqlWf0&ab_channel=UnrealCG
  http://martiancraft.com/blog/2015/02/disintegrating-baddies/

Time estimate and required duration / problems encountered

In advance I wanted to take about a full working week for the project, more if necessary.

Ultimately, I (roughly adjusted, since this requirement was not known from the beginning) needed approximately the following amount of time:

After 2 intensive days, I had already implemented most of what I wanted to do roughly using starter content and blueprints.
Another day took me to create the level, to find suitable assets, so that everything looks harmonious and good, the distances fit, etc… Actually, I originally wanted to use Jorge Rodrigues' 3D model (listed in external content), which is why I contacted him. He even made his model available to me free of charge, unfortunately the integration took me a lot of time and in the end didn't lead to anything: First I had to reduce the polygons several times in Maya for the almost 1.5 GB model, but then the texture has no longer fit and I haven't found out why and at the end of the day I haven't been able to create animations for it. A friend kindly agreed to create a grouse model for me, which I could import directly into UE and add animations using mixamo. The whole procedure took me another day.

The conversion of the initially temporarily built-in string append into a ListView, as well as the repeated start of a disintegration shader took another day, but unfortunately none of these features work as planned.

A few hours were then spread over several days in polishing, such as postprocessing effects, adding sound effects and music, adjustments in the menu and level, as well as clean-up work such as packing in functions and using custom events for the blueprints.

After the last lecture I used another day to familiarize myself with the C ++ implementation and added a utility class for sorting a key-value dictionary.