Callum Ferguson

Battle Royale Tank Game

Github: https://github.com/CallumFerguson/CS4610FinalSpring2020

Youtube: https://www.youtube.com/watch?v=qxom37T8zKA

Description of the System Developed

My final project is a game developed using WebGL. Since WebGL is only a graphics library, I had to both create a game engine and make a game using this engine. The primary purpose of the game engine I built is to keep the WebGL code separate from the game logic. This means this project is effectively two projects. One which is the engine that handles rendering, scene graph, and game assets (shaders, models, textures, etc), and one which handles the world generation, player movement, and multiplayer logic.

The actual game is a multiplayer battle royale tank game. Any number of people can join by going to the game url. Before the game starts, you can attack other players, but you will respawn. Once the match starts, all players will be teleported to a random position in the map. A

sand storm will shrink in size causing the players to move towards the middle. Players who die will not respawn. The last player alive wins.

Methods Used

Rendering:

For the rendering, most of the code is similar to what we did in class for our homework assignments except that it was converted to javascript and WebGL. The engine creates objects that store the VAO and any data specific to an object such as its position. Every frame, it loops through all of the objects and their children, and it draws each one by binding its VAO, binding its shader program and texture, and sending any uniform variables to the shader.

Scene Graph:

The engine supports having a scene graph which means objects can have children that move in relation to their parents. An example where this is used in the game is the barrel of the tank is a child of the rotating turret, and the turret is a child of the tank. This means that when I rotate the turret, the barrel also rotates. To achieve this, each object's model matrix is multiplied by all of its parent's matrices. The engine handles this automatically in an efficient way where the matrices are only calculated when there are changes to an object's position, rotation, or scale.

Multiplayer:

I used socket.io for the multiplayer. The server is very basic. It only relays the information about the position of each player if they shoot. Each client takes the information about other players and renders a tank in the position of each player. I have a separate client for connecting to and controlling the server. This allows me to reset and start a match.

Terrain Generation:

The world is generated automatically using a procedural mesh for the ground, then it places trees and walls. The generation is based on a seed that the server generates and sends to the clients when they connect. This way all of the clients generate the same random map. The terrain mesh is created by making a grid of points that are connected with triangles, then the height is controlled by several layers of 2d simplex noise. Then the normals are calculated the same was as in assingment3b.

Game Play:

When players connect, they are all in the “waiting for host to start match” state. Once the host (me) sends the command to the server to start, the server tells all of the players the match has started. Each player is teleported to a random position on the map, and a sand storm which is just a large cylinder with an animated texture begins to shrink. If a player is killed by another player or goes into the sand storm, they tell the server they have died. All players remove that player’s tank model from the map. Then the player enters the “spectator” state where they can fly around and view the rest of the match.

Sand Storm:

The sandstorm is a large cylinder surrounding the map that shrinks in size over the course of the match. It is created with a special shader that uses several layers of 3d simplex noise. It uses the uv texture coordinates of the mesh, but instead of using a texture, it uses the uv coordinates and time into the 3d simplex noise. By using time as the third dimension, the texture is smoothly animated so it looks like a sand storm.

Remaining Issues

Most of the remaining issues are small bugs which could be fixed relatively easily if I had the time. One major issue, however, is the physics of the game. The physics of the tanks and the networking is very unrealistic and easy to break. While some of the problems can be fixed by me, many of the problems are because of the physics engine I choose to use. I choose a lightweight easy to use physics engine to save myself time and to keep the file sizes low. The physics engine, Cannon.js, is not the most accurate physics engine, but more importantly, it is missing certain features. The main two I needed was the ability to detect continuous collisions between objects and the ability to use ray casts into heightfields. Without either of these features, there is no simple and reliable way to detect if the tank is on the ground. This means the player can continue to control the tank while it is in the air. A potential solution is to constantly check the tanks y position and compare it to the values produced by the noise that

creates the ground, but a better solution would have been to use a better physics engine. I did not have enough time to change the physics engine my game engine used.