Games Architecture 600098 Report

Scott White-2022149404

Chapter 1 Game Engine Design and Critique

Chapter 2 Evaluation

High Score Client Server Implementation

Within the game there is a server which is used to store high scores of the players which have played the game. This is set up in a client server configuration which means that each device will connect to the server itself and get the scores from the server. When the game is first loaded up there is contact made with this server to extract all high scores which are saved on it. Then once you complete the game you contact the server to send your initials and score to be saved on the server. The way that this server was implemented was that it uses a TCP connection, this means that you can connect to the server using the same IP address as the computer as they are run on the same device, you then must set the port that it is using to make sure they communicate with each other. It then writes and reads strings which are being sent between the client and the server causing things to happen, such as sending a get string to the server means it will retrieve the high scores, and sending any string other than get to the server will save that string as a high score. This required me to do networking code which I am not as comfortable with, so to help me complete this task I used GitHub copilot, this is a way of generating code by using comments in the code and then it will suggest code for you to use. I used a combination of this and looking at the code in the networking lab, for example I typed code to send and receive high scores and then it generated the networking code to allow me to do that, I then changed it fit more of my needs and work with the rest of my code.

AI Algorithms

Within the game there are three AI algorithms that I have used, these are used to control the enemies in my game which are the drone and the rolling object.The main driving force behind all these AI algorithms is a velocity, this allows for them to move by setting a vector, I can then use steering forces which means I can take a position that I want the entity to go towards and make the velocity vector move towards that by finding the difference in positions, this is used for all the AI algorithms that my obstacles use.

The first algorithm that I have implemented is a roaming algorithm, this means that an entity will move around a set area to random points within that area. The way that this works, is that I have created entities with only positions components, and then added a component that means they are recognized as roaming points, this then means that all entities with this component will be added to a list. From this list a random point is selected, then the entity which is roaming will be steered towards that random point, once it has reached that point, it will look for another random point and then go towards that. This works quite effectively however there is a chance that the random point selected is the same point as the one that was selected before meaning that it would stay in the same place, another issue with this approach to roaming is that it means it can only go to certain points in the selected area, when it would be more ideal if it could go to any point within a selected radius. The reason I created a roaming algorithm is that I thought it would be ideal for obstacle which rolls around on the floor.

The next algorithm that I have created is a pathfinding algorithm, this is a way for an entity to finds the direction to a point, by having different nodes it can go to and then finding the shortest route by looking at the weighting. This was implemented by using a basic a\* algorithm, this means that there are nodes and a target, you calculate each weight for each node, by looking at the distance to that node, and then distance from that node to the target, it would then select the node with the least weighted distance and then would keep doing this until it reached the target. The way that I have implemented this in the game is that I have created entities to act as the nodes which have a position component, and a component to indicate that it is a node, It then looks through each node, and if they are close enough to the entity controlled by the AI, it sees them as neighbouring nodes, it then uses the A\* algorithm with all the nodes that it sees as neighbours and then steers based on that. One of the issues with my pathfinding algorithms is that the nodes don’t have set neighbouring nodes, and instead rely on distance between nodes to determine when something is a neighbour, by having set neighbouring nodes, this means that you could use it over a larger area and wouldn’t have to keep adjusting the neighbour range value to make it fit the needs. The pathfinding algorithm was used for the drone, this because it needs to move around the whole maze without going through the walls, and by finding the best path to the player, which meant that this AI algorithm was ideal.

Game Production Problems

While making this game there was lots of issues I had to solve when it came to programming, due to doing new things that I haven’t had to do before and programming in the ECS way. One of these issues that I encountered was when trying to get a way to type the initials of a player when registering a high score, this is because I was struggling to find a way of seeing whenever any key is pressed, the way I sorted this was I figured out that you can check to see whether a key is being pressed by using an index as well a keycode, this meant that I could look at each key index to see when any is being pressed and then add that to the initial string, in order to allow player to add their initials, this then had issues as it kept adding the letter more than once when pressing it, this meant I had to store when a key was pressed in a variable so that if it registers that key as being down again, it wont add it to the initial string until you release that key, and press it again.

Another issues I encountered while programming was that I couldn’t figure out how to set up the pathfinding algorithm for the drone, this is because I could get it to come towards the player, but it wasn’t avoiding the walls, which is not the behaviour that I wanted, this meant I had to change the approach and algorithm that I was using the pathfinding by switching to a basic A\* algorithm, this wasn’t working at first as it was going back and forth between the same nodes, this made me realize I needed to set a current node, and then calculate the weighted distance from that node instead of from the drone itself. This fixed some of the issues as the drone was now following the shortest path to the player however, it was still travelling through walls, as the nodes with the lowest weighted distance weren’t neighbour nodes, this meant that I had to find a way to distinguish neighbouring nodes, I did this by if they were a certain distance away this would mean that it was a neighbouring node, this solved all the issues I was having and now meant the drone was pathfinding towards the player without going through walls anymore.