Illuvium Test

NOTE: This repo only contains the source files, solution file, uproject and contents. Git wasn't allowing me to push such a large amount of data. For full version of game, you can download it from here: https://drive.google.com/drive/folders/1l8xat5JGowoJYPNiT1eddrbzsGlA0dh4?usp=share_link

1. Introduction

I made this game using Unreal Engine 4.24. I have mainly used C++ to achieve my goals. I only used blueprints to create red and blue balls' prefabs.

I have used **UE_LOG** warning to make messages stand out, as they are yellow in color.

I did spend a little more than 6 hours on this task :)

2. What I made

These were the sub tasks that were expected from this test and how I did it:

1. *Server and client (replica) system where every thing happens on the server and the client simply replicates it and show it on the screen.*

   Answer: My code has 3 main class: Client, Server and CoreServer. All the simulation happens on the CoreServer. The updates on the CoreServer are done by the Server class. These Client class will read data from the server and perform any visual update. The Ball class represents the players on the screen and the blueprints are made using it.

2. *The server has to be headless i.e. it should be able to run without a client. This means that even if we remove the client, the server should run without any issue. So the server is not reading any game data from client (it will however be running TCP/IP and sockets which will require communication from the client to establish and maintain the connection)*

   Answer: The Server + CoreServer can run on their own. The just provide const public functions that the Client can read from. But the Server + CoreServer do not read anything from the Client.

3. *The server should run the update after every100 ms. So even if the client is running 60 fps, it cannot receive any new update until the next 100 ms.*

   Answer: I have fixed the delta time to 100 ms for the CoreServer. It runs a while loop and accumulates time until it reaches 100 ms. It is then that it performs the update by calling the CoreServer::UpdateState() function.

4. *The client provides a scene with a hex grid (I changed it to square grid for simplicity). The*

players will be walking on the discrete grid cells i.e. the player cannot be standing on 2 or more tiles at once. It can only do that when traversing but once it is done moving it should stop at a discrete cell.

Answer: The grid is made up of square tiles. And the client reads the new locations of players from the server. Once read, it linearly interpolates them to new location by calling Client::LerpBall() function. The grid is spawned by the Client by calling the function Client::SpawnGrid().

5. *There are 2 balls red and blue. These balls will find each other on the grid, traverse to each other and then fight until one of them dies.*

   Answer: I created the red ball and blue ball using blue print. They are spawned by the Client by calling Client::SpawnBalls(). The location is read from the server as to where the balls will be placed.

6. *The player locations and hit points are randomly generated. However, it should be deterministic in the generation of those random numbers.*

   Answer: All the random numbers are generated by FrandomStream. The seed is a private member in the CoreServer clas called the m_RandomStreamSeed.

7. *There should be a visual cue for letting the viewer know the health of the players.*

   Answer: I wanted to make the material of the balls glow when they take damage. However, I realized that this will change the material instance which will affect all the other balls that are using the same material (like in 2 red vs 2 blue). I decided to add a point light over the top of the balls. Their intensities would decrease when they take hi. But it wasn't clear enough. So I added a small sphere over each ball. The scale of this small sphere decrease as the player takes hit.

8. *The player hit should happen within a certain number of frames.*

   Answer: The players frame data for hitting is set to 2. This number can be changed by changing the CoreServer::m_AttackFrame. I made is this so that there is an anticipation frame and then the hit frame.

9. *The players should lerp to their location.*

   Answer: The players move to their location by linearly interpolating. The Client::LerpBall() function handles lerping. I made my custom lerping function so that it can be easily changed to allow different interpolations like quadratic or cubic interpolations.

10. *A path finding algorithm should be used.*

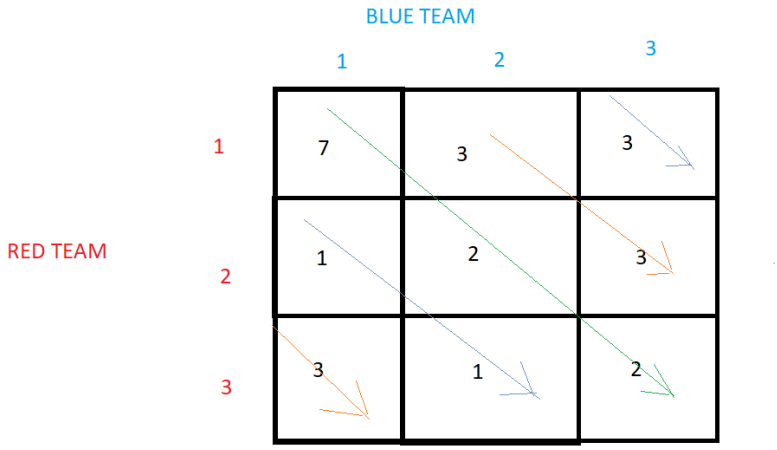
    Answer: I used A* path finding. Since it is a heuristic search, I used the Manhattan distance for

calculating distances.

3. Things I couldn't add

1. *The game can allow more than 1 player i.e. 2 red and 2 blue etc.*

   Answer: I designed the game to allow 2 v 2. However, I couldn't complete it. From the beginning I was taking 2 v 2 into account but due to lack of time I couldn't complete it. The game flow would have to be changed. A state machine might be better used here. The players would have to find the closest enemy. This was a hurdle since an enemy can be closest to both the players. Greedy algorithm might have been used to find the closest enemies (since the sum of all players to their enemies has to be smallest). Like in the image below where the purple arrow shows the best combination for each player and its enemy:



4. Bugs

There is a visual bug where the small health sphere and the point light on each player does not follow them. You need to click each ball once so they update the location of point light and health ball properly. Secondly, the point light intensity is updated but it is not rendered. Again the player ball has to be clicked to get the proper intensities of the point lights.

Although I have added log messages in the CoreServer class that can provide much better information

about the simulation.

5. Things that would have made it interesting

I was trying to add obstacles to the game. But it was a little too much work. I had to leave it incomplete . There is a parameter on the Grid class called m_IsWalkable. This parameter was meant to serve this purpose.

6. Video

I have added a video showing the working game. The game is reset after one simulation to show that it is deterministic. The video is in the Video folder and is called: ***Illuvium_1Redv1Blue.mp4***