CMP303 - Networking

Keiran Millar - 1502338

The Application

- The game was intended to be a multiplayer version of brick breaker, where two players would have to protect their blocks from a ball that players would bounce between the paddles that they control.
- So far I have been able to implement the players paddles and send this between the server and the other client, with prediction for the other player.

Network Architecture: Client - Server

- I chose to do a client to peer network due to the application only being run on the one system, so there isn’t any latency due to the distance from client to server and back.
- It also the variables for the application are stored on the server so it stops clients from being able to tamper with the application.
- The player will send their position to the server and the server will then save this information and send it to the other client.
- The server also has a state machine which changes dependant on whether clients are playing the game or waiting for another player to connect.

Application-Layer Protocols

- In my application I am using SFML to help make the application look a lot better. I have a paddles class which I make an array of to help keep track of variables, I also have useful functions in this class for setting variables and getting the body for displaying.
- When a user connects to the server I will send them their place in the list of players (currently 0 and 1 but if another player connects they will get 2) and then they will wait until the server has told them to start the game, this allows for player 0 to wait for another person to join, stopping there game from crashing.
- After two players have connected the server then sends a message to everyone telling them to start playing the main application.

Application-Layer Protocols

- Before a message is sent I ensure to send a string that describes what information is about to be sent, for example I send “playerInfo” before I send the server the id of the user and their new positions, ensuring the message is processed properly.
- When the client receives the position of the other player, they will save the location the other client was at to their previous position and will use this alongside the position they have received to predict where the other player is going to be. There is also a Lerp function I created to help create a smoother transition between the different positions of the users

Networking API: TCP

- TCP was used for this application as I knew that there would be two people playing the game at one time.
- This API is also very reliable and will deliver a packet in the correct order
- This advantage helps to keep the variables consistent and not get the X and Y co-ordinates mixed up by accident.
- If a packet is lost then TCP will resend a packet to compensate for this error.
- It is connection based so a connection is made at the start of the program to ensure all users are connected

Code Structure: Event Based IO

- My code uses an event based IO, this allows the application to only send information when something has changed, for example my program will only send a players position if it has changed since the last update.
- This helps keep message transferral to a minimum.

Prediction

- I have used linear prediction due to it being closer to the true location of the other player when it does go wrong.
- Also since the application is only two dimensional it will be fairly straightforward for this to work, when it does go wrong it is also closer to the correct position than if I was to implement a quadratic formula for the prediction.
- If I was to implement a ball for the paddles to hit it would be simple to allow this prediction to be used for that as well since it can move in the same directions as the user’s paddles

Critical Discussion

- A few ways that my application is vulnerable is that there is no way for the server to handle more than 2 connections, this would cause a third clients game to send information to the server that is never processed and it would be unplayable.
- The client-server setup would also be hugely hindered if it was to be spread further apart which would result in a lot of latency between users.
- The application hasn't had a feature implemented yet that would close the game for a user if the other client disconnects, this results in the user controlling their paddle and the other paddle will be stationary since nobody will be on the other side to control it.

Additional Wishes

- Given more time I would have implemented a ball that the users could hit between them and this would require more precise prediction and interpolation to give a more believable collision detection
- I would like to have added more states for my gamestate variable, this would allow for the game to be paused and for there to be a starting screen in the application.
- I would like to implement the before mentioned feature of closing the game for the other player if a client disconnects.
- I would like to let other clients to connect and be able to "spectate" the two players playing the game.
- My paddles already have a function that allows for the previous positions of the paddle to be saved and the previous 3 were used for a calculation but I couldn't get it working reliably in the application, although it is never used I would have liked to get it working.