ASGENetGame Developer Diary

AN INTERACTIVE CONNECT FOUR EXPERIENCE – OLIVER HEWISON

Day 0 - Research

For this assignment, my partner Andrei and I began by discussing what we wanted to make. We discussed which board games we had played and enjoyed throughout our lives, and we came to decide that we would attempt to recreate an accessible game from our childhoods, adding features to make it even more enjoyable.

Eventually, we decided on Connect Four, an easily grasped game that could be extended further. We looked at existing variations of the game together:

- **PopOut** – instead of inserting a counter, a player may choose to pop out one of their counters from the bottom row of the board, dropping all counters in the column by one. This adds a significant amount of depth and strategy to the previously simple game, while retaining its easiness to learn.
- **Connect Five** – The game is instead played on a 9x6 grid as opposed to 7x6. This makes the game lengthier but easier. From our playtesting, this was not as interesting as the default game, but it should be easy to implement a variable board size.
- **Power Up** – This gamemode is much more complex, and adds "Power Checkers" which when placed, influence the board, for example, an anvil piece pops out all the pieces below, effectively resetting that column. This would be much more complex to implement, but we thought it could be a stretch goal.

Figure 1: A physical Connect 4 board with variation possibilities for many game-modes at once.

Week 1 - Planning & Preparation

We then began thinking about how we would implement the game in C++. This was my first time doing any kind of networking or multi-threading work, so first I spent some time getting to grips with the concept of client-server networking.

We then made a brief development timeline, outlining what we would like to achieve at certain points in time, planning to work on a proper networking and game framework first before working on the game so we would save time in the long term and have re-usable code assets for future projects.

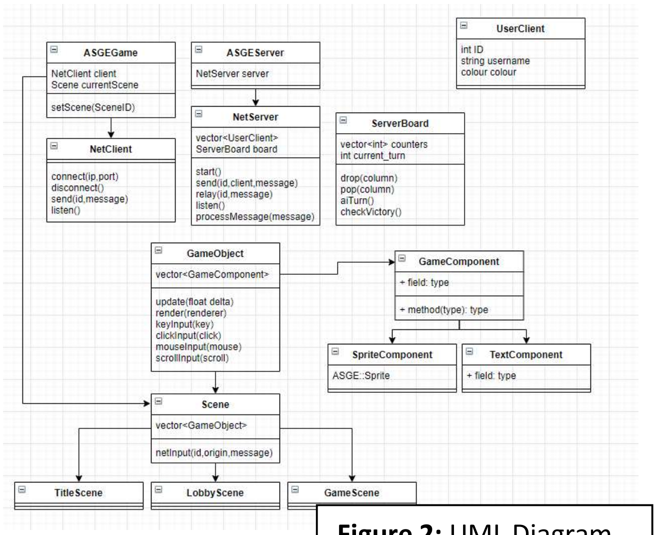

Figure 2: UML Diagram

Development

Phase I – Building the Framework

For the first few weeks of development, we split up our handling of the networking, myself handling the sending of messages from client to server, and Andrei handling relaying of the messages back to the client. I then got to work on implementing a scene system that allowed us to split up parts of the game into different classes, allowing us to work effectively without overlap.

Whilst Andrei put together the Lobby, Game and Win screens, I worked on putting together my version of a netstring, that allows command IDs to be passed between client and server, which the server then adds on the origin socket's player ID, a unique identifier that is allocated to the player on connection.

I also added a callback function to the client, which allows a NetInput function in the current scene to be called with the interpreted netstring, allowing for changes to be made across the network.

Phase II – Putting the Game Together

With a lot of the baseplate code handled between Andrei and myself, I then began work on a Title screen, allowing the players to type in the IP address, port and their username before the client connects.

In this time, Andrei made a variable-sized board display in the Game Scene, so after, I worked on sending moves between players, using the server as a relay. I also added a Chat Window Game Object, which allows players of the game to have a chat room in which they can discuss anything they'd like to, such as comparing game strategies.

We then had a working version of the game, though most of the processing is done client-side as opposed to server-side. We decided for our personal options to pick AI and Disconnection/Reconnection support, so we would have to move a lot of this server-side, which I spent a lot of time on for a few weeks, also working on a victory detection algorithm.

Phase III – Option Choices

Now that we had the game to a playable standard, it was time to implement our option choices, mine being AI. To prepare, I improved the player turn algorithm so that if only one player is connected to a lobby, an AI takes over.

I then thought about how I would want my connect four AI to work. The base Connect Four gamemode has previously been mathematically solved, meaning that a perfect AI could never lose, and algorithms already existed on the internet to implement it in that way.

I, however, wanted a non-perfect AI that tries to act like a regular player, so I ended up programming it in such a way that it will try to win if it is close to completion, and will attempt to prevent the other player from winning, but otherwise it will attempt to stack its pieces conveniently.

To improve the player experience, I deliberately allowed the AI to ignore if a player is stacking its pieces in a way that there will be no way for the AI to recover, which focuses on what myself and Andrei found to be the most fun when playtesting.

Post-Mortem

In retrospect for both of our first times using networking or multithreading, I am happy with what we have achieved and learned. We have completed a fully playable connect four game with additional features like chat rooms, AI and disconnect/reconnect support. We would have liked to have pushed the server-side networking a little further, including a working lobby system that allows for the creation and joining of games, however that would have been too much to expect from our first time networking.

Certain bugs arose from multithreading, particularly the fact that there is a small probability that when the win condition is met, two threads attempt to access the a class at the same time which would have required more time than we had towards the end to fix, which would have been done using mutexes.

I am, however, looking forward to using more multithreading in future games, both singleplayer and networked. It allows for much more performant code, especially where otherwise I would have used an update loop to achieve a task such as the counters dropping.