

CE303 Advanced Programming

Assignment 1

Thomas Bloom 1601217

Contents

Running the server and clients	3
Command line	3
Clicking the Jar files	4
Program architecture	5
Package Structure	5
Semi-Modular	5
Handling concurrency	5
How networking was handled	5
Server Class:	5
Player Class:	5
GameState Class:	6
User interface design	6
Frame (Window)	6
Grid	7
Bottom Panel	7
Project review	7
How the project went	7
What was challenging	7
Unfinished features	7
Review of project management	8
What I would do next time	8

Running the server and clients

Command line

If you are already in the project folder, open the build folder. Hold shift and right click inside the folder and select "Open with PowerShell window". Then simply type in this syntax

java -jar <FILENAME>.jar

So, for the server it would be...

java -jar Server.jar

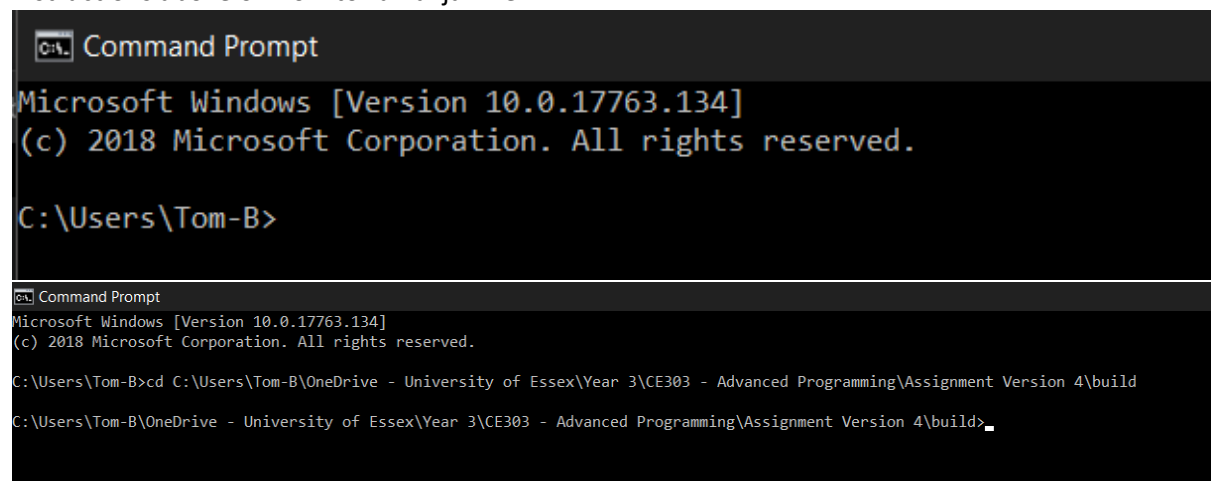
And for the client...

java -jar Client.jar

If you are not in the project folder, simply open command prompt or PowerShell and type

Cd <FILE PATH TO BUILD FOLDER>

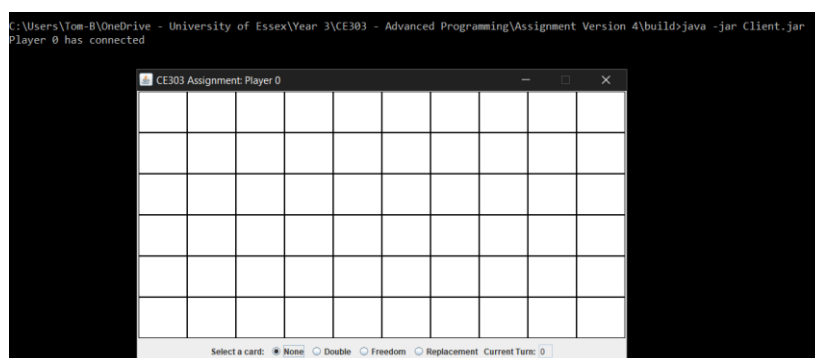
This will change the directory your command window is looking at, you can then follow the instructions above on how to run a .jar file.



```
C:\Users\Tom-B>

C:\Users\Tom-B>cd C:\Users\Tom-B\OneDrive - University of Essex\Year 3\CE303 - Advanced Programming\Assignment Version 4\build
C:\Users\Tom-B\OneDrive - University of Essex\Year 3\CE303 - Advanced Programming\Assignment Version 4\build>

C:\Users\Tom-B\OneDrive - University of Essex\Year 3\CE303 - Advanced Programming\Assignment Version 4\build>java -jar Server.jar
Server is now running
```







Clicking the Jar files

From the project folder, navigate to the build folder. You will see two .JAR files one called “Server” and one called “Client”.

First double click on Server.jar, this will open a small window that will have text showing that the server is now running.

Next, double click on Client.jar, this will open the window in which you can play the game. If you are the only player in the game then the grid will be empty, but when the other player joins, the grid will be populated with the starting colours.

The other player can then navigate to the same Client.jar file and run it, they will be added to the game, so now both players are connected and can play.

University of Essex > Year 3 > CE303 - Advanced Programming > Assignment Version 4 > build					
Name	Status	Date modified	Type	Size	
 Client.jar		29/11/2018 14:58	Executable Jar File	15 KB	
 Server.jar		29/11/2018 14:58	Executable Jar File	15 KB	

Program architecture

Package Structure

From the start of the project I wanted the structure of the classes to be very organised, so I decided that I would group classes that perform a similar function or complement each other in the same package. This made it a lot easier to switch between classes during development because they were already right next to each-other in the project view.

- Server (Server, client, Player)
- User interface (Cell, window)
- Utilities (Coordinate, Game state)

Semi-Modular

Instead of simply putting all code relating to the user's end of the program into one class, I split it up into multiple classes (Client and window). The purpose of this was to make the code a lot more manageable so I didn't have lots of code all in one class. An advantage I found by doing things this way was that it was much quicker to make changes as I was able to get to the part of the code that I wanted to change quicker, instead of having to go through a long class to get to what I wanted.

Handling concurrency

In order to make sure that my program didn't come into any issues with concurrency, I made sure that the server managed whose turn it is. If it is not that player's turn then they cannot have any affect on the game, thus nothing is being accessed at the same time by the two players.

I also made the method that checks if the move is legal synchronized, this ensures that it is thread safe.

How networking was handled

Server Class:

I implemented networking into the game by using sockets. The server has a ServerSocket object set to the same port as the players. It then constantly checks to see if a player has connected, if it has then a Player object is created (if a player does not connect then the object is not created, and the game does not start). The Player objects are then added to a list and constantly iterated over, during each iteration the player is sent a message. This message will hold information on the position and colour of the cell that has been clicked on.

Player Class:

The player class acts as a bridge between the client and the server. It has a reference to the accepted ServerSocket, keeps track of the player number and also has a reference to the server's gamestate object (this is where the server keeps its game board).

When the Player object is first created (when a player joins the game) it issues a command to the client with a welcome message, the client then receives this message and changes the title of their window to reflect their player number.

The logic to handle if a move is legal is in this class, I chose to put it here because it is only accessed in the Player class, so it made sense to keep it all together in the same class. This method takes in a Coordinate object as a parameter, this holds the position of the cell that it is about to check. It first checks to see that it is the current player's turn, if it is then it moves onto the next check. It then checks to make sure the cell is empty (no player has already placed their colour on it before). After that it checks all the surrounding cells to make sure that there is a cell of that player's colour next to it. If there is a cell directly next to it of the same colour, then the method returns true because a cell can be placed here. If any of these checks fail, then the method returns false and the player must choose another cell to click.

This class also contains arguably the most important method for the networking side of this project. The 'SendMessage' method. This simply checks to see if the server is sending out a "LEGAL" message (which means that the player is allowed to place their colour in that cell), if it is then it sends the same message onto the client to reflect the changes on the client's grid. Without this method, the client wouldn't be able to receive any messages from the server.

As this class extends Thread, it needs a run method. The purpose of this method is to take in commands from the client, check them and if it passes the legal move checks then it sends the command to the server. It does the opposite of that the sendMessage method does.

GameState Class:

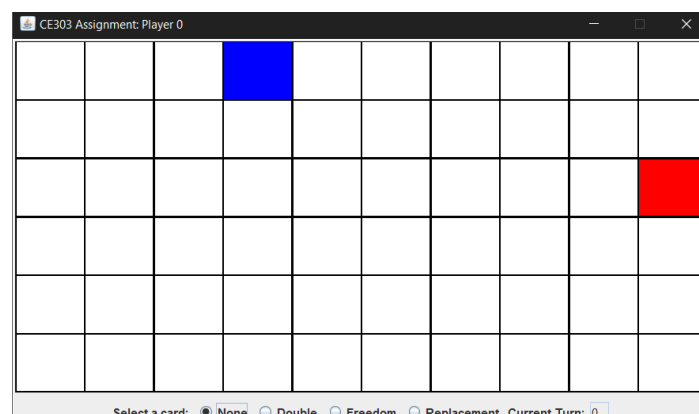
This class handles the current state of the server, there is only one instance of this class (which is in the Server class). It contains information such as the grid size, the message being sent to the client, which player's turn it is and has its own board. In the constructor it creates its own board and makes it so the cells are not owned by either player. When a player moves, the cell is updated. It also contains public methods that are called in other classes to get and set its variables.

User interface design

Frame (Window)

This is the window that the player will see on their screen, I aimed to make it as simple as possible. The window is a fixed size that cannot be resized, if you resize the width and not the height then the grid would become distorted, so I fixed the size so that the grid's cells stay as squares.

The title of the window is changed when the player enters the game to show which player number they are, I felt that this was an easy solution to showing the player their number and it also ensures that they can see it as the game is always in windowed mode.



Grid

The grid takes up most of the window as it is the most important part of the program. The cells are clearly separated with black borders, so any player can clearly see each cell. Each cell is a square so that the grid doesn't look distorted.

Bottom Panel

The panel at the bottom doesn't take up much room as it only consists of text and buttons. The elements in the panel are centred because it looks much better than if they were left-aligned. This is where the player would be able to select their influence card, while I didn't have enough time to implement these, I am glad I managed to sort out the user interface. The influence card buttons are grouped together to ensure that only one button can be selected at a time, this just makes sure that a player cannot have multiple cards active at the same time.

Project review

How the project went

I found this project to be very difficult because I personally struggle with networking and understanding how it works, and it just so happened that most of this project is based on networking.

I found the user interface to be the easiest part for me, I had a grid that you could click on up and running quite quickly but syncing it up with more than one player was the hard bit, more on that in the next section.

But overall, I am very proud of myself for doing as much as I did, I (mostly) overcame the problems that I faced with networking, so I feel that my knowledge of networking has increased drastically. I feel that if I had more time to work on the project I would have finished with a more complete project.

What was challenging

I found the networking side to be extremely difficult. It took most of the development time to just get it up and running, which left me with very little time at the end to implement the other features (influence cards, bots, end state). I couldn't wrap my head around how-to setup networking in the first place, then understanding how to constantly get and receive messages between the client and the server.

Unfinished features

Unfortunately, I didn't have enough time to fully implement the win/lose condition into the game. I did work on it for a bit, but I didn't have enough time to get it working. A player should lose the game if they get blocked in by the other player so that they cannot make any more moves. While I do have checks for the surrounding cells, I wasn't able to tweak this to check if there are any available cells to place on, it only performs these checks on a cell that is clicked, which the player obviously wouldn't be able to do if they were blocked in.

Another unfinished feature is the influence card system. I managed to implement the user interface for this, but I didn't have much time to work on the logic behind each card. The double card works but there is a known bug that causes the card to not work if it isn't your turn. Each message sent

does have information on the current card being used. Instead, when an influence button is pressed, the button disappears so the user cannot click it again. This is an easy way of ensuring each player can only use influence cards once.

I didn't make a start on the bot as once again I didn't have enough time to get started on it, as most of my time was spent tackling problems with networking. This would have also been a very time-consuming area of the project for me because I have never worked with AI before, so I wouldn't have had enough time to work on it.

The counter to show whose turn it is isn't working correctly, I couldn't find out why it wasn't updating because I was updating the variable that the text field was showing. If I had more time to work on the product then I would be able to solve this problem.

Review of project management

As you have gathered from the previous section, my time management wasn't good for this project. Juggling many assignments that were all due around the same time was the root cause of this, but I cannot blame my poor time management completely on that.

I am pleased with myself for pushing through problems that I faced, working long hours to ensure that I had a product to show. While I wish I didn't have to work long hours at the end of the project, I feel that doing so helped me to stay focused.

I am also very proud of the way that I laid out the project, it made development a lot easier because I didn't have to spend much (if any) time at all when trying to find a certain piece of code. Splitting up the project into packages made it so much easier to manage, it is something I will do for my future projects.

What I would do next time

I feel that the best thing I can do for next time is to start the project much earlier than I did for this one as many of the features for this project were not implemented because I ran out of time. For example, I couldn't fully implement the influence cards because I reached the deadline for the project, however I was able to make a start on them.

Another thing that I will need to do next time is to ask for help earlier if I get stuck, unfortunately I left it until the week of the deadline to seek help with this work, which didn't leave me with much time to do the actual coding. Next time, along with starting earlier, I must have more courage to simply ask for help. However, my problem is that because of anxiety I find it much more difficult to ask for help. I was only able to get help this time from my personal tutor because I had a meeting with him anyway.