cjd47
University of Bath

CM10228: "Dungeon of Dooom" - Coursework 2: Adding a GUI (Java):

Specification and Test Plan

I will be extending the code submitted in CM10228 coursework 1 to develop a basic GUI along with 2 advanced features from the specification presented to us.

The two advanced features that I have chosen to implement are Advanced Features 1 and 3.

Whilst I will endeavour to make the look window update even when a player isn't moving, this will probably be done through 'client pulling' instead of 'server pushing', hence I have decided against pursuing Advanced Feature 2.

Ideally, mine allow will allow users to connect to the client code that produced in Assignment 1. To connect with other people's servers, I will provide editable text fields so that the user can enter IP addresses and port numbers for connecting to them.

If more than one person is playing in the dungeon, or an AI is also playing in the dungeon, then events might happen without the user doing anything. As such, the player panel should auto-update often enough for changes to be reflected across all clients in real-time.

The **basic requirement** of the coursework is to **build a GUI for human users**. The must:

- **Operate in a window**
- **Have a control panel consisting of buttons**
- **Have a pane that shows the user the outcome of their actions.**

These are the requirements that must be met. I will therefore definitely be using labels, a large text pane for showing feedback from the server, and a number of buttons to control the player in the dungeon or quit the application.

On top of the basic GUI I will implement the following two advanced features:

Advanced Feature 1:

"Add a **graphic pane** to your Player GUI to **display what is going on** without resorting to scrolling text.

• **Update** this pane for **each player move**

• **Update** this pane for **each bot move**

• **Show** player/bot **progress** towards winning the game"

Advanced Feature 3:

"Create **another GUI** (the Game Engine GUI) for the **game engine** when it's a **server** (requires client / server architecture).

• Create a **"god's eye view"** pane that lets you watch everything that happens in the dungeon.

• Report the **IP address** for your **server**.

• Use **radio buttons** to turn the **actual serving for clients** (the listening) **on and off**. Allow a **new port to be selected**, but disable this if the server is currently serving / listening"

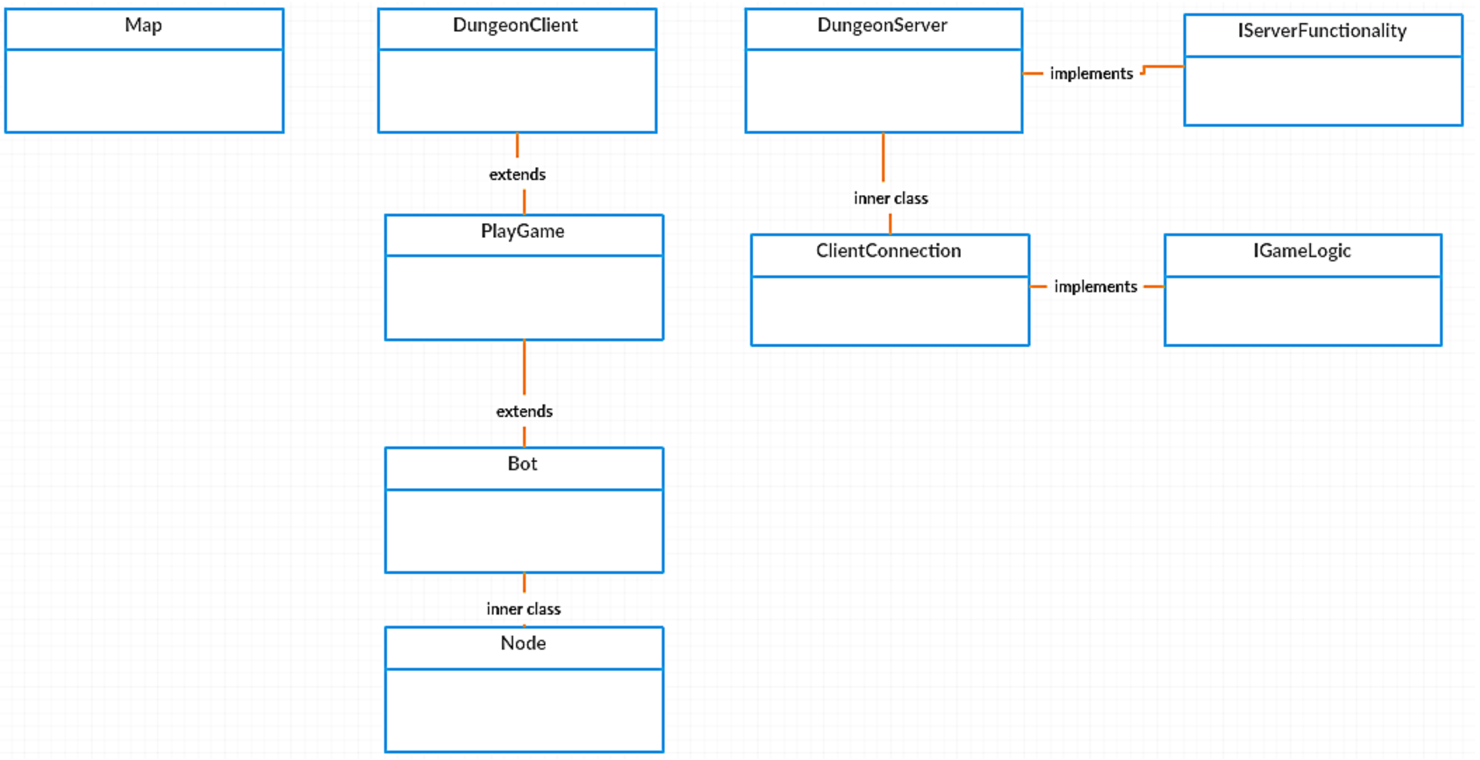
NB: In this case, I will take "listening" to mean that a client's requests are met by a server response. In other words, if the radio button is selected, the client requesting to move in a certain direction or pick up a piece of gold will be handled by the server. If the radio button is de-selected (the server is NOT listening), the request will be ignored.

High Level Design

Class structure

Original architecture (in Assignment 1)

I will start off with the following architecture in coursework 2. It is the class structure that was submitted in the first assignment. This is the code that I will start off working with in Assignment 2



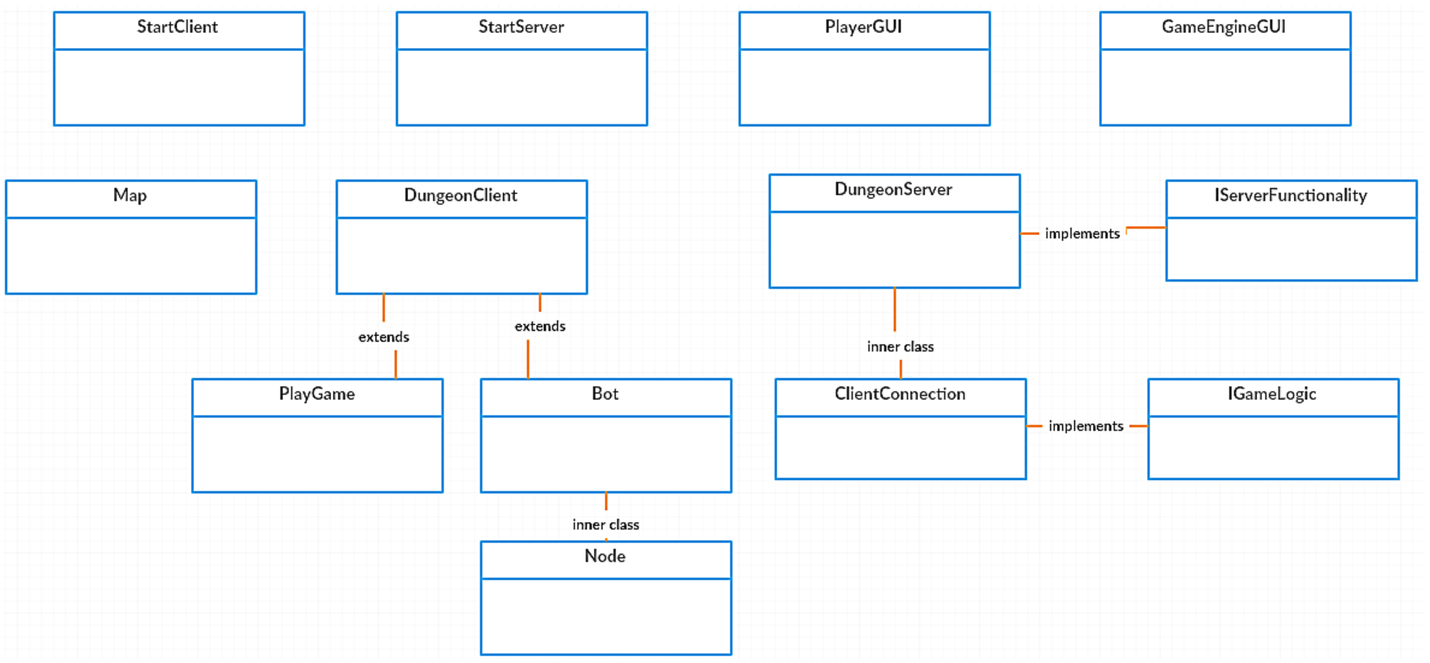
Proposed architecture (in Assignment 2)

In assignment 2, I intend on developing the class structure to **accommodate for 4 GUI classes** to be added:

- *PlayerGUI.java*
- *GameEngineGUI.java*
- *StartClient.java*
- *StartServer.java*

I will also change the structure slightly, so that **Bot.java extends DungeonClient directly**. This is so that the Bot can directly influence the GUI instead of accessing any of PlayGame's attributes unnecessarily.

This is the architecture I envisage I will end up with after coursework 2 is complete.



Low Level Design

GUIs to implement

1. **StartServer.java:**

Purpose: This GUI will be used to start the server. It will allow a user to specify a port number between 0 and 65536 (2^{16}), type this number into the editable text field shown and then run a server which launches using that port number. A new port can be selected, but this is not allowed if the server is currently serving / listening.

Description of class: This class will be called first out of all classes because it is used as an interface to start the server (without the server running, a client cannot connect to any running game). It will contain a main method which builds the GUI responsible for choosing a server port. It will create a JFrame with a BorderLayout consisting of 4 JPanels (top, left, middle, right). The game logo will be in the top panel and the components responsible for choosing the port and launching the server (JTextField and JButton) will be placed in the middle panel. The action listener for the "Start Server" button will validate the text entry to check if a valid port number has been selected, before creating a new instance of DungeonServer.java, passing the chosen port number into its constructor so a server can be launched using that port number. Upon invalid entry, a JOptionPane displaying an error message will appear.

Logo: Yes – centre of Top Panel (Page Start)

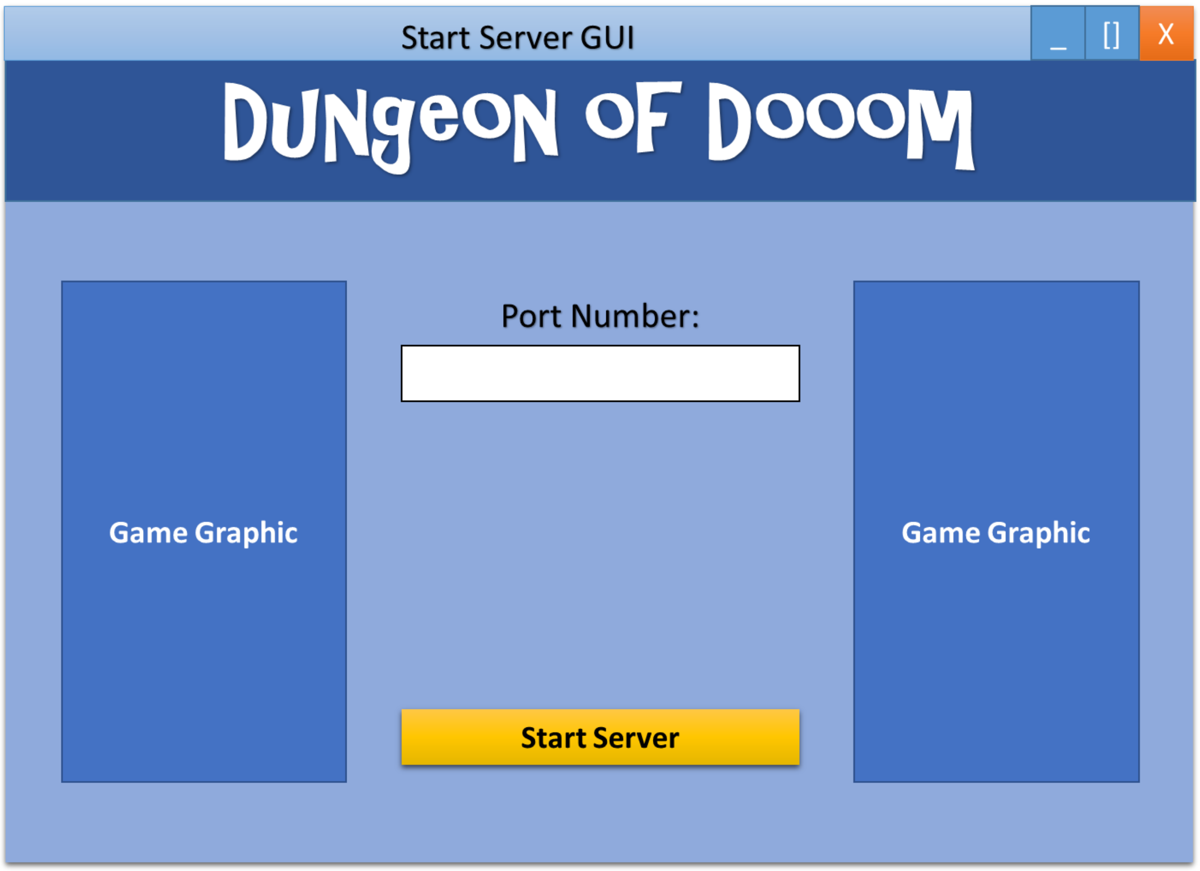
Font: Lucida Console

Layout Managers: BorderLayout (Page Start, Line Start, Centre, Line End)

Components: JFrame, ImageIcon, JLabel, JButton, JTextField, JOptionPane

On-close behaviour: DISPOSE_ON_CLOSE

Sketch:



2. **GameEngineGUI.java:**

Purpose: This GUI will be for the game engine when it's a server (requires client / server architecture). It will contain a "god's eye view" pane that lets you watch everything that happens in the dungeon. This is unlike the player's view, which only shows what the player can know. This GUI also reports the IP address of the server. There will be a radio button to turn the actual serving for clients (the listening) on and off.

Description of class: This class will be called upon instantiating the DungeonServer class. An instance of GameEngineGUI will be created in the DungeonServer constructor. From there, a method will be called to build the GUI for the server. GameEngineGUI will contain a method which builds the GUI and other methods to update the "God's eye view" and print to the text area. The buildGUI method will create a JFrame with a BorderLayout consisting of 4 JPanels (top, left, middle, right). The game logo will be in the top panel and the components responsible for displaying server IP address/client connections, printing the "god's eye view" of the map and buttons to shut the server down will be in the left and right panels respectively. A JRadioButton will be used to determine whether or not the server listens to or ignores client requests. This will be done by checking if the button is selected before calling the ParseCommand method.

NB: In this case, I will take "listening" to mean that a client's requests are met by a server response. In other words, if the radio button is selected, the client requesting to move in a certain direction or pick up a piece of gold will be handled by the server. If the radio button is de-selected (the server is NOT listening), the request will be ignored.

Logo: Yes – centre of Top Panel (Page Start)

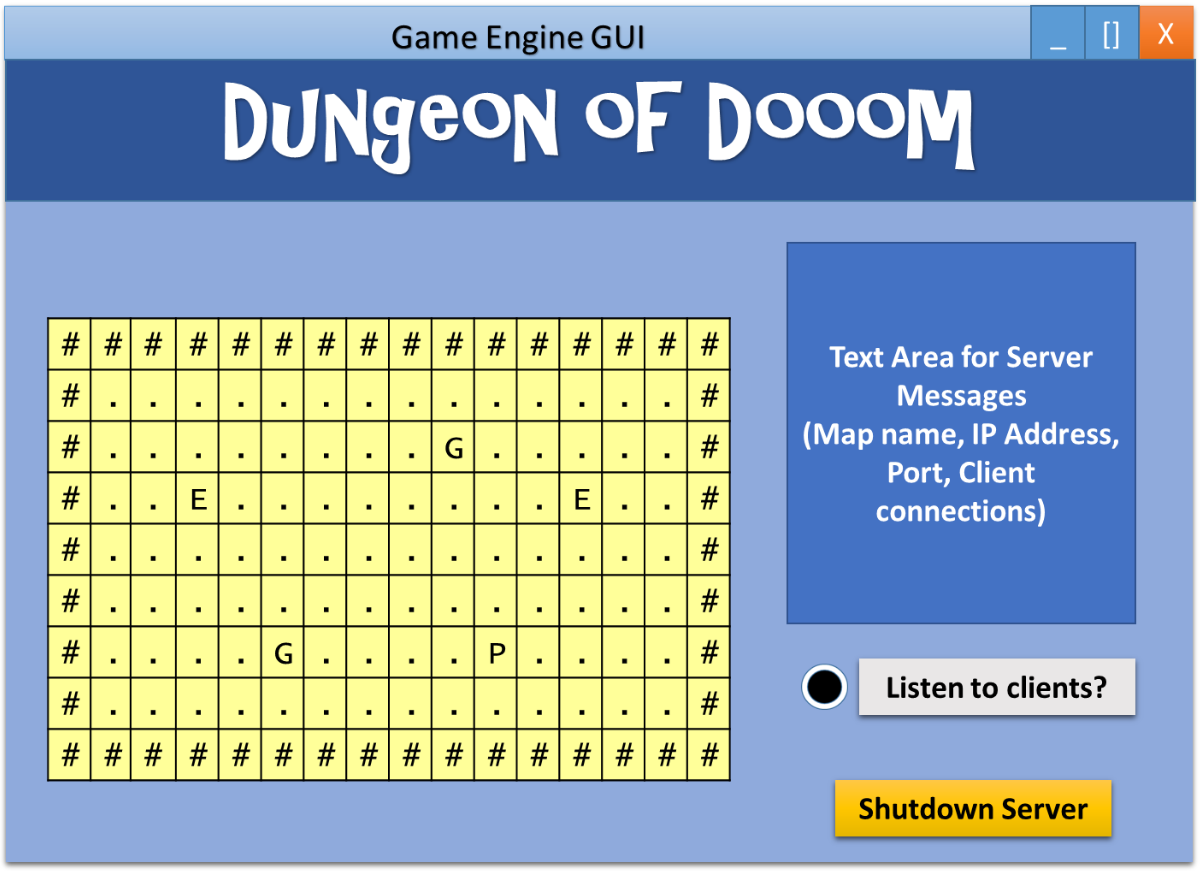
Font: Lucida Console

Layout Managers: BorderLayout (Page Start, Line Start, Centre, Line End)

Components: JFrame, ImageIcon, JLabel, JButton, JTextArea, JRadioButton

On-close behaviour: DISPOSE_ON_CLOSE

Sketch:



```
# # # # # # # # # # # # # # # #
# . . . . . . . . . . . . . . #
# . . . . . . . . G . . . . . #
# . . E . . . . . . . . E . . #
# . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . #
# . . . . G . . . . P . . . . #
# . . . . . . . . . . . . . . #
# # # # # # # # # # # # # # # #
```

Text Area for Server Messages (Map name, IP Address, Port, Client connections)

3. **StartClient.GUI:**

Purpose: This GUI will be used to launch the client (human player or bot). It will include editable text fields for the IP address and port number allowing any valid combination of the two to be entered so that a client can connect to any server. This GUI will have two buttons, one launching the human player and the other launching the bot. It will have a similar design to the StartServer GUI.

Description of class: This class will be called second out of all classes by the user because it is used as an interface to start the client (either bot or human player). It requires a server to already be running so that a client may connect to a running game. It will contain a main method which builds the GUI responsible for choosing a port and IP address to connect to. It will create a JFrame with a BorderLayout consisting of 4 JPanels (top, left, middle, right). The game logo will be in the top panel and the components responsible for choosing the port and IP address and launching the client (JTextField and JButton) will be placed in the middle panel. The action listeners for the "Start game" buttons (for both human and bot players) will validate text entry to check if a valid port number has been selected, before creating a new instance of PlayGame.java or Bot.java, passing the chosen port number and IP address into its constructor so a client can be launched using those credentials. Upon invalid entry, a JOptionPane displaying an error message will appear.

Logo: Yes – centre of Top Panel (Page Start)

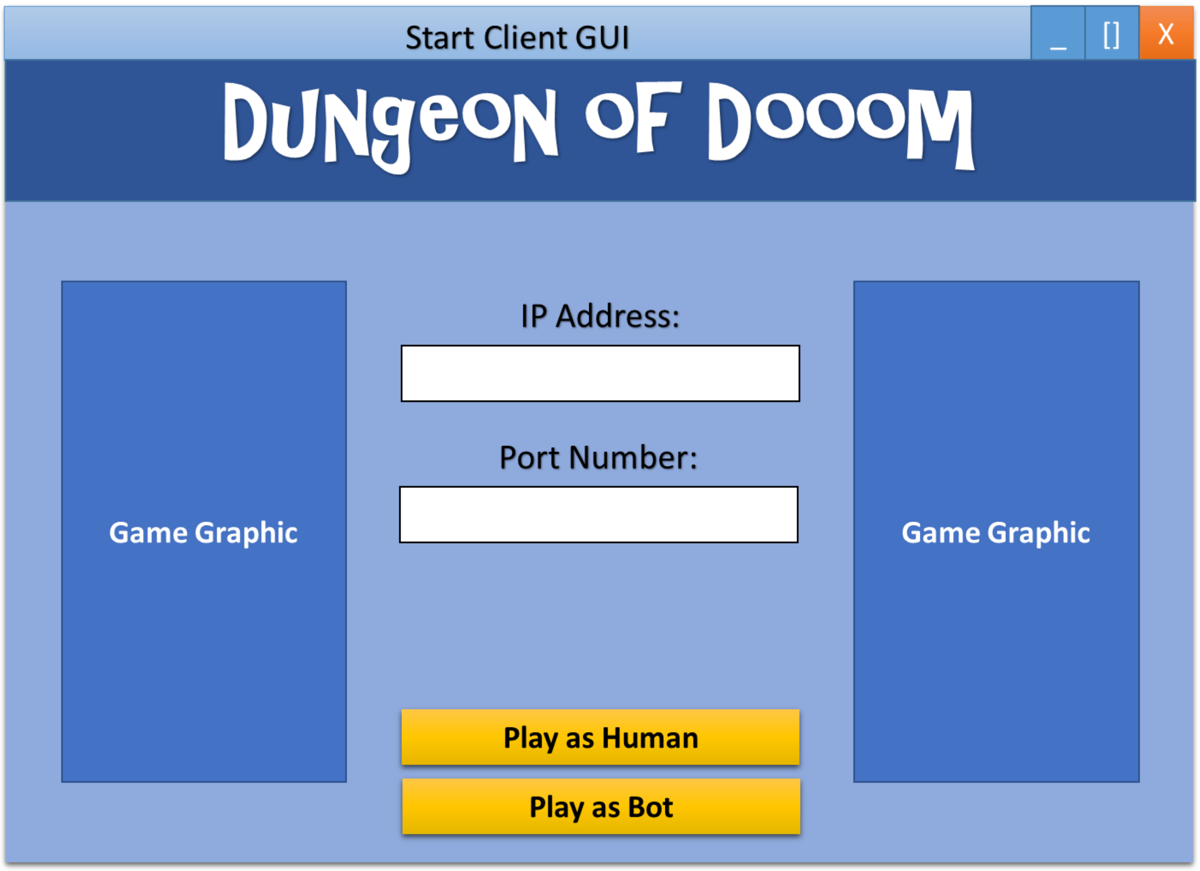
Font: Lucida Console

Layout Managers: BorderLayout (Page Start, Line Start, Centre, Line End)

Components: JFrame, ImageIcon, JLabel, JButton, JTextField, JOptionPane

On-close behaviour: DISPOSE_ON_CLOSE

Sketch:



4. **PlayerGUI:**

Purpose: This GUI will have labels to explain commands/for display purposes, a large pane for showing feedback from the server via a scrolling text area, and a number of buttons for controlling the player movement through the dungeon, picking up gold as and when it needs to. A graphic pane to will display what is going on in the look window around the current player. This will update for each player move and each bot move. Finally, this GUI will also show player/bot progress towards winning the game.

Description of class: This class will be called upon instantiating the DungeonClient class. An instance of PlayerGUI will be created in the DungeonClient constructor. From there, a method will be called to build the GUI for the player (client). PlayerGUI will contain a method which builds the GUI and other methods to update the "look window" for the player and print server feedback to the text area after each command. The buildGUI method will create a JFrame with a BorderLayout consisting of 4 JPanels (top, left, middle, right). The game logo will be in the top panel and the components responsible for displaying server feedback, printing the "look window" of the map and buttons to control the player in the dungeon will be in the left, middle and right panels respectively. The look window will update after each player move and automatically by calling the "LOOK" command in regular intervals so clients always see the updated window. This is true for human players and bots. A **count for the gold remaining for a given player to escape the dungeon** will be shown in a client's window (on the PlayerGUI). This will show player progress towards winning a game, along with recording the moves made in the text area. This means after a single player has collected a piece of gold, their tally will decrease whilst others stay the same (as the amount of gold others need to escape is unchanhed).

Logo: Yes – centre of Top Panel (Page Start)

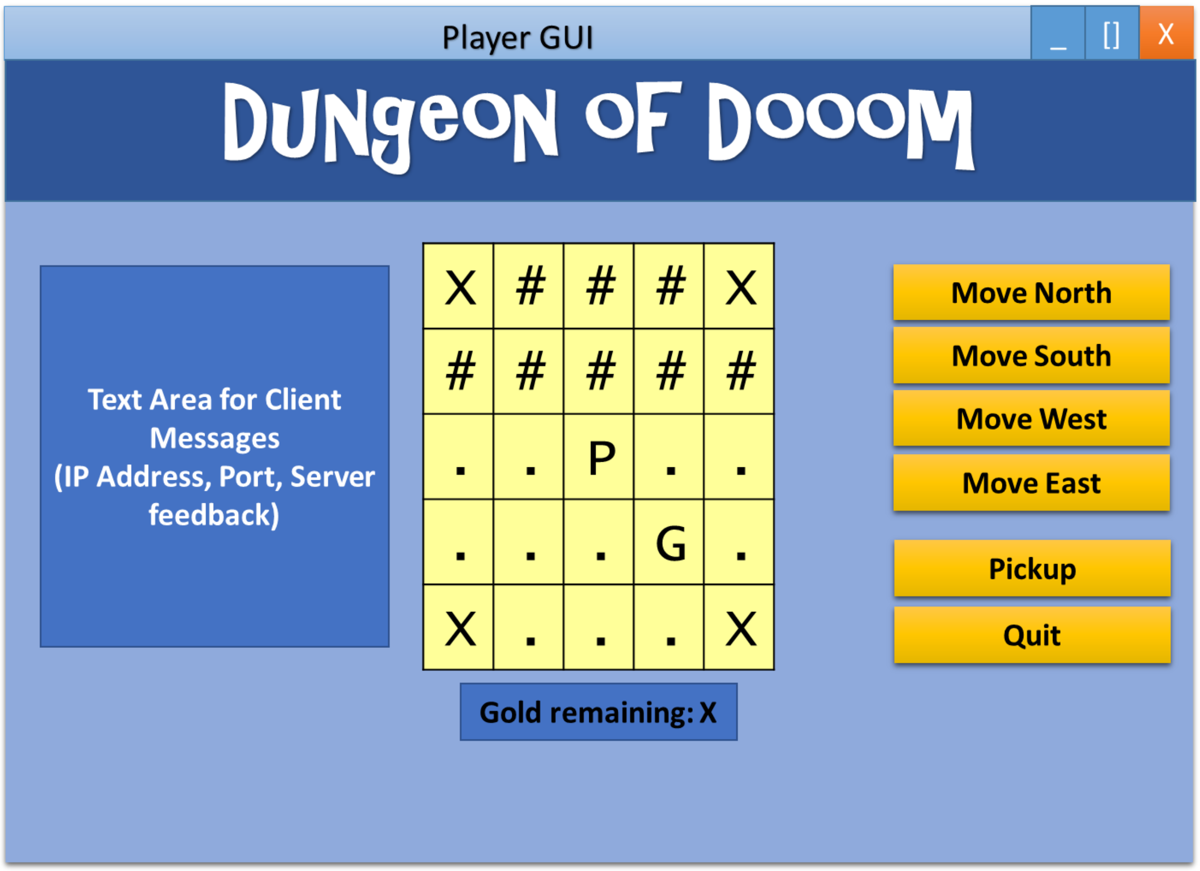
Font: Lucida Console

Layout Managers: BorderLayout (Page Start, Line Start, Centre, Line End)

Components: JFrame, ImageIcon, JLabel, JButton, JTextField, JOptionPane

On-close behaviour: DISPOSE_ON_CLOSE

Sketch:



Algorithms

1. **PlayerGUI will use an algorithm as below to print the player look window**

```
public printLookWindow (<char[]> lookWindow) // pass in ArrayList of char arrays
    remove all components from panel
    set grid layout
    set constraints and dimensions for 5x5 grid

    create new 5x5 char array lookWindowArray[5][5]

    for i=0 to 4 {
        for j=0 to 4 {
            set lookWindowArray[i][j] = lookWindow.get(i)[j]

        }

    }

    for i=0 to 4 {
        for j=0 to 4 {

            if lookWindowArray[i][j] contains '#' wall tile
                place wall tile image in position i,j

            else if lookWindowArray[i][j] contains '.' empty tile
                place empty tile image in position i,j
                else if lookWindowArray[i][j] contains 'P' player tile

                if i=2 and j=2
                    if player is a human
                        place human player image in position i,j

                    if player is a bot

                        place bot player image in position i,j

                else

                    place other player image in position i,j

                else if lookWindowArray[i][j] contains 'G' gold tile
                    place gold tile image in position i,j

                else if lookWindowArray[i][j] contains 'E' exit tile
                    place exit tile image in position i,j

                else if lookWindowArray[i][j] contains 'X' edge tile
                    place edge tile image in position i,j

            }

            refresh panel containing grid to update player view

        }
```

2. **GameEngineGUI will use an algorithm as below to print the entire map**

```
public printMap (char[][] wholeMap, int height, int width)
    remove all components from panel
    set grid layout
    set constraints and dimensions for width x height map grid

    for i=0 to height {
        for j=0 to width {

            if lookWindowArray[i][j] contains '#' wall tile
                place wall tile image in position i,j

            else if lookWindowArray[i][j] contains '.' empty tile
                place empty tile image in position i,j

            else if lookWindowArray[i][j] contains 'P' player tile
                place player image in position i,j

            else if lookWindowArray[i][j] contains 'G' gold tile
```

```
                place gold tile image in position i,j

            else if lookWindowArray[i][j] contains 'E' exit tile
                place exit tile image in position i,j

            else if lookWindowArray[i][j] contains 'X' edge tile
                place edge tile image in position i,j

        }

        refresh panel containing grid to update player view

    }
```

3. **Bot.java** will use a **Breadth First Algorithm** to determine the shortest path to its target (if there is gold to collect then that target is gold. If there is no more gold to collect than that target is the exit).

   "Breadth-first search assigns two values to each vertex v:

   A distance, giving the minimum number of edges in any path from the source vertex to vertex v.

   The predecessor vertex of v along some shortest path from the source vertex. The source vertex's predecessor is some special value, such as null, indicating that it has no predecessor.

   If there is no path from the source vertex to vertex v, then v's distance is infinite and its predecessor has the same special value as the source's predecessor."

   (Source: https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/the-breadth-first-search-algorithm)

Functional Requirements

Index	Name	Description	Priority
1	**Build a GUI responsible for launching the server**	This GUI should be created with the vision of allowing a user to choose a port for the server to run on, before launching that sever on the designated port at the click of a button.	**High**
1.1	**Ensure the server can be launched on any chosen port**	It will allow a user to specify a port number, type this number into the editable text field shown and then run a server which launches using that port number.	**Moderate**
1.1.1	**Ensure chosen port is a valid port number**	It will allow a user to specify a port number between 0 and 65536 (2^{16}). Otherwise, an error message should be displayed.	**Moderate**

1.2	**Allow a new port to be selected, but disable this if the server is currently serving / listening.**	If a server on a given IP address is currently running on a given port, it should not be possible to launch another server on the same port. Also, whilst a game is running, it should not be possible for the port number of the running server to be changed.	**Moderate**
1.3	**Ensure the StartServer GUI operates in a window, includes control buttons and an editable text field**	The GUI responsible for launching the server will run in its own window, before opening a new window when a valid port number has been entered and the user has clicked a button starting the server on that port.	**High**
2	**Build a GUI responsible for launching the client**	This GUI should be created with the vision of allowing a user to choose an IP address and port number combination for the client to connect to a running server on, before launching that player (human or bot) by opening their player GUI at the click of a button.	**High**
2.1	**Ensure this StartClient GUI can launch both a bot and a human player**	This GUI will have two buttons, one launching the human player and the other launching the bot.	**High**
2.2	**Ensure this StartClient GUI can connect a client with any chosen running server with a chosen IP address and port number**	It will include editable text fields for the IP address and port number allowing any valid combination of the two to be entered so that a client can connect to any server.	**Moderate**
2.3	**Ensure the chosen IP address is valid**	It will allow a user to specify an IP address of either "localhost" or of the form "xxx.xxx.xxx.xxx". Otherwise, an error message should be displayed.	**High**
2.4	**Ensure the chosen port number is valid**	It will allow a user to specify a port number between 0 and 65536 (2^{16}). Otherwise, an error message should be displayed.	**High**
2.5	**Ensure the StartClient GUI operates in a window, includes control buttons and editable text fields**	The GUI responsible for launching the server will run in its own window, before opening a new window when a valid IP address and port number combination has been entered and the user has clicked a button connecting the client (human or bot) to the server on that port.	**High**

3	**Build a GUI responsible for running the game engine when it's a server**	This GUI will represent the server. It will be responsible for reporting on its IP address and printing messages when new clients connect or existing clients disconnect. It will also show the map as a "God's eye view" displaying everything on the map, including clients in their respective positions.	**High**
3.1	**Ensure the GUI reports the IP address of the server**	The server's IP address is reported in the text area on the GUI along with the map name and all client connection activity.	**Moderate**
3.2	**Ensure the GUI contains a graphical "God's eye view" of the map with players overlaid on it**	This lets the user watch everything that happens in the dungeon. This is unlike the player's view, which only shows what the player can know.	**High**
3.3	**Ensure the GUI contains a radio button to turn the actual serving for clients (the listening) on and off**	There will be a radio button to turn the actual serving for clients (the listening) on and off. This means that when a client makes a request to move or pickup, that request is not greeted with a response if the radio button is not set to 'selected'. Conversely, all requests are greeted with a response if the radio button is set to 'selected'. **NB: In this case, I will take "listening" to mean that a client's requests are met by a server response. In other words, if the radio button is selected, the client requesting to move in a certain direction or pick up a piece of gold will be handled by the server. If the radio button is de-selected (the server is NOT listening), the request will be ignored.**	**Moderate**
3.4	**Ensure the GUI reports when a client connects and disconnects**	A text area will exist that reports when clients connect to, and disconnect from, the server.	**Low**
3.5	**Ensure the GUI includes a button to shut down the server and close the socket**	A "shutdown" button will be responsible for closing the server. This will close the server socket and exit the system gracefully, disconnecting all clients.	**High**
3.6	**Ensure the GameEngineGUI operates in a window and includes control buttons**	The GUI responsible for running the game engine (server) will run in its own window. Control buttons will exist to shut the server down and to turn listening on / off.	**High**

4	**Build a GUI responsible for running the client when it's a human player**	The GUI will have a player view representing what a player sees in the dungeon. This will include their look window and how far that specific player is away from escaping the dungeon.	**High**
4.1	**Ensure this GUI also work for bot players**	The GUI will be adapted (there will be less buttons as actions are automated) for bot players. Moves and updates to the look window graphic pane will be made automatically.	**High**
4.2	**Ensure multiple players can play on the same map with separate windows for each**	Each new client will be launched in a separate window after they have connected to a server. This is so that multiple clients may play using the same machine.	**High**
4.3	**Ensure there is a graphic pane to display what is going on without resorting to scrolling text**	A graphic pane to will display what is going on in the look window around the current player.	**High**
4.3.1	**Ensure the graphic pane updates for each player move**	Every move that is made will be proceeded by a "LOOK" call, which updates the graphic pane.	**High**
4.3.2	**Ensure the graphic pane updates for each bot move**	Every move that is made will be proceeded by a "LOOK" call, which updates the graphic pane.	**High**
4.4	**Ensure labels are used to make it clear how to use the application**	This GUI will have labels to explain commands – it will explain to the user what pressing each button does.	**Moderate**
4.5	**Ensure control buttons exist to let a player move and pick up gold in the dungeon**	A number of buttons for controlling the human player's movement through the dungeon will exist (these will be hidden for bot players). Bot players will pick up gold automatically once they are on top of a gold tile.	**High**
4.6	**Ensure to include a quit button which closes the client's connection**	A quit button will be displayed for both human and bot players. These will allow the client to disconnect from the game.	**High**

4.7	**Ensure to include a pane that shows the user the outcome of their actions.**	There will be a large pane for showing feedback from the server via a scrolling text area, accompanied by the graphical user window.	**High**
4.8	**Ensure the PlayerGUI operates in a window and includes control buttons**	The GUI responsible for showing the client will run in its own window. Control buttons will exist to control movement (for human players) and allow the client to disconnect from the server.	**High**
4.9	**The look window should update automatically, even if a move is not made**	There will be a routine call of “LOOK” which is made in regular intervals that updates each player’s graphical pane regardless of whether they have made a move or not.	**Low**
4.9.1	**Player progress towards winning a game should be shown through a progress bar / gold count**	Finally, this GUI will also show player/bot progress towards winning the game by providing a progress bar and corresponding count for the number of gold coins remaining until that specific player can escape from the dungeon.	**Moderate**

Non-Functional Requirements

Index	Name	Description	Priority
1	**Must be written in the Java OOP language.**	The chosen language for the project is Java - an Object Oriented language which seems most appropriate for the development of a GUI as it uses objects based on classes.	**High**
2	**Must expand upon the initial requirements.**	Requirements will be expanded upon from those set out in the “basic GUI” section of the brief, to ensure the program includes advanced features.	**Low**
3	**Must have scope for further extension/integration.**	The project, when finished, should be designed in such a way that there is plenty of scope for further improvements. These improvements may be the extension of the project to run on multiple platforms etc.	**Moderate**
4	**Must have an aesthetically pleasing user interface.**	It is paramount that the user interface is aesthetically pleasing, as well as intuitive to use. Maintaining an MVC model whilst designing the UI is important. This allows the interface to be considered separately from the main functionality of the software model.	**Moderate**

5	**Must accommodate users of a range of IT literacy - intermediate to expert.**	By simplifying the UI so that users with a broad IT literacy can use it, this will extend the possible number of users of the software and make it more accessible to new users. That said, users must have sufficient IT knowledge from having played similar simple computer games before.	**Moderate**
6	**Must be compatible on any Windows machine capable of running the appropriate version of Java.**	Due to the system not running on multiple platforms, the user must initially have a machine capable of running the software - in this case it will be a Windows machine capable of running the appropriate version of Java.	**High**
7	**The software will not infringe on any existing copyrights, trademarks or patents.**	All sources used in creating the documentation or used to aid the software development will be appropriately credited/cited/referenced.	**Moderate**
8	**The software must make use of a high contrast of colours**	A high contrast of colours must be used in the software in order for users with poor vision to be able to read text	**Low**
9	**Code structure should be clear and concise. Comments and Javadoc where necessary will aid the clarity of design.**	It is important that, not only the GUIs function to their purpose, but also that the way they have been coded is clear and concise. All code should be clearly abstracted and commented.	**High**
10	**The program should scale well for multiple clients**	As the game can be played by multiple clients, it is important that the GUIs are scalable, meaning that each client can have their own client window.	**Moderate**

Test Plan

The "Success" column would has been filled out after the program was built and testing took place.

Unit testing

StartServer GUI			
Test No	**Description**	**Expected Result**	**Success? (Yes/No)**
1	**Start server button** Clicking "start server" after a port number between 0 and 2^{16} has been entered in the port number field.	On the click of this button, if validation tests are passed, the server is launched on the specified port (GameEngineGUI opens on that port, displaying the chosen number as its port).	✓

2	**Port number field validation** Ensure the port number is an integer between 0 and 2^{16}. Validation starts when the Start Server button is pressed. Valid entry = 40004 Invalid entry = 400000000004	When Start Server is pressed, numbers between 0 and 2^{16} such as 40004 are accepted (GameEngineGUI opens on that port, displaying the chosen number as its port), whereas numbers outside these limits such as 400000000004, or data of incorrect format ('aaaaa' or '42.31') are greeted with an error message.	✓
StartClient GUI			
Test No	**Description**	**Expected Result**	**Success? (Yes/No)**
3	**Start human player button** Clicking "start human" after a port number between 0 and 2^{16} and a valid IP address has been entered in each field.	On the click of this button, if validation tests are passed, the human player is launched into a game on the specified port and IP address (PlayerGUI opens in a game on that port/IP address, displaying the chosen number as its connected port).	✓
4	**Start bot player button** Clicking "start bot" after a port number between 0 and 2^{16} and a valid IP address has been entered in each field.	On the click of this button, if validation tests are passed, the bot player is launched into a game on the specified port and IP address (PlayerGUI opens in a game on that port/IP address, displaying the chosen number as its connected port).	✓
5	**Port number field validation** Ensure the port number is an integer between 0 and 2^{16}. Validation starts when the Start Server button is pressed. Valid entry = 40004 Invalid entry = 400000000004	When Start Human or Start Bot is pressed, numbers between 0 and 2^{16} such as 40004 are accepted (PlayerGUI opens on that port, displaying the chosen number as its port), whereas numbers outside these limits such as 400000000004, or data of incorrect format ('aaaaa' or '42.31') are greeted with an error message.	✓
6	**IP address field validation** Ensure the IP address is of the format "xxx.xxx.xxx.xxx" or "localhost". Validation starts when the Start Human or Start Bot button is pressed. Valid entry = localhost Invalid entry = 343455.565656.2332334.11111	When Start Human or Start Bot is pressed, addresses of the form "xxx.xxx.xxx.xxx" or "localhost" such as "192.068.123.001" are accepted (PlayerGUI opens on that port, displaying the chosen address as its IP address), whereas data outside these limits such as 343455.565656.2332334.11111, or data of incorrect format ('aaaaa' or '42.31') are greeted with an error message.	✓
GameEngineGUI GUI			
Test No	**Description**	**Expected Result**	**Success? (Yes/No)**

6	**Shut down server button** Clicking the "shutdown" button whilst a server is running.	The GameEngineGUI closes and the server socket also closes cleanly. If any clients are currently connected, they are notified that the server socket has closed. Any further button presses are not met by server responses.	✓
7	**"God's eye view" grid** The map is loaded correctly and updated after each client move.	When the GameEngineGUI loads, a map with no players is loaded, which updates when a client connects or any player moves.	✓
8	**"Listen to clients" radio button** The selection and de-selection of this radio button determines whether requests are responded to by the server. **NB: In this case, I will take "listening" to mean that a client's requests are met by a server response. In other words, if the radio button is selected, the client requesting to move in a certain direction or pick up a piece of gold will be handled by the server. If the radio button is de-selected (the server is NOT listening), the request will be ignored.**	When the radio button is selected, client requests are met by a response – players can move around and pick up gold. When the radio button is de-selected, new clients can join but the requests they make to move around the dungeon and pick up gold are not met with a response.	✓
9	**IP address of server reported** The IP address of the server is printed when the GameEngineGUI loads.	When the GameEngineGUI loads, the IP address of the server is fetched and displayed in the text area.	✓
10	**New client connects** A new client connects to the running game.	The text area updates and a message is appended saying the new client has connected, along with its client ID.	✓
11	**Client disconnects** An existing client disconnects from the running game.	The text area updates and a message is appended saying the client has disconnected, along with its client ID.	✓
PlayerGUI GUI			
Test No	**Description**	**Expected Result**	**Success? (Yes/No)**
12	**Move north button** Click on the Move North button.	The player moves north by one square. Its position changes, the look window graphical pane changes to reflect the objects around it in this new position, and the "God's eye view" updates with the player in its new position. The text pane on the client side prints "SUCCESS" when a move is made.	✓
13	**Move south button** Click on the Move South button.	The player moves south by one square. Its position changes, the look window graphical pane changes to reflect the objects around it in this new position, and the "God's eye view" updates with the player in its new position. The text	✓

		pane on the client side prints “SUCCESS” when a move is made.	
14	**Move west button** Click on the Move West button.	The player moves west by one square. Its position changes, the look window graphical pane changes to reflect the objects around it in this new position, and the “God’s eye view” updates with the player in its new position. The text pane on the client side prints “SUCCESS” when a move is made.	✓
15	**Move east button** Click on the Move East button.	The player moves east by one square. Its position changes, the look window graphical pane changes to reflect the objects around it in this new position, and the “God’s eye view” updates with the player in its new position. The text pane on the client side prints “SUCCESS” when a move is made.	✓
16	**Pickup button** Click on the Pickup button.	The player picks up the gold in the current square. When the player moves away, the previously-gold tile is replaced by an empty tile in the look window and “God’s eye view” updates. The text pane on the client side prints “SUCCESS” if a gold coin is collected and “FAIL” if no gold is collected. If a piece of gold is collected, the progress bar and tally of gold remaining per client should update.	✓
17	**ID button** Click on the ID button.	The ID of the current player is printed to the text area on the client side.	✓
18	**Quit button** Click on the quit button.	The PlayerGUI closes and the client’s connection is removed. This disconnection is shown in the ‘client activity’ text pane on the GameEngineGUI.	✓
19	**Game won** Check the conditions for when a game is won by a certain player.	When a player has collected all necessary gold to win a game, (when their “Gold to win” tally = 0), they move to an exit tile. Once they click “Move” in the appropriate direction, this takes them to the exit tile, closes the server and displays a win message to the winning client, disabling their control buttons.	✓

Automated testing (JUnit)

```
/**

These JUnit tests will ensure whether data can be written to and read from the text areas
on both GUIs. This is a crucial part of both GUIs so that the client can print server
feedback and the server can display its IP address as well as report on client
connections.

When tested, the tests came back successful ✓

*/

import static org.junit.Assert.assertEquals;

import org.junit.Test;


/**

 * Automatic testing.

 * @author cjd47

 */

public class JUnitTesting {


	public static void main(String[] args) {

		JUnitTesting tests = new JUnitTesting();

		tests.runTests();

	}


	public void runTests(){

		GameEngineGUI ge_GUI = new GameEngineGUI();

		PlayerGUI pl_GUI = new PlayerGUI();


		ge_GUI.textArea.setText("Game Engine");

		assertEquals("Game Engine", ge_GUI.textArea.getText());

		ge_GUI.textArea.setText("Player");

		assertEquals("Player", ge_GUI.textArea.getText());

	}

}
```

Feature Table

Intended Feature	Sub-Features	Max Points Available (From Specification)	Self-Evaluation (does feature work?)	Special Instructions
Basic Player GUI (“PlayerGUI.java”)	• Operate in a window • Have a control panel consisting of buttons • Have a pane that shows the user the outcome of their actions.	**20**	**Yes**	This GUI is loaded **after a server has been loaded** using StartServer, and it is **launched using StartClient.java.**
Advanced Feature 1: Add graphic pane to Player GUI to display what is going on without resorting to scrolling text **(PlayerGUI.java)**	• Update this pane for each player move • Update this pane for each bot move • Show player/bot progress towards winning the game	**25**	**Yes**	This graphic pane updates after every client move and **also every 500ms.** It works by the client calling “LOOK” regularly (it works on a client ‘pulling’ basis). This is so that the updated window is shown even when a player is not moving. **On some occasions, this causes multiple clients running on a single machine to have windows overlapping one another when it updates and brings itself to the front of the screen.** Progress is shown through a “gold to win” tally and progress bar.
Advanced Feature 3: Create another GUI (Game Engine GUI) for the game engine when it's a server. **(GameEngineGUI.java)**	• Create a "god's eye view" pane that lets you watch everything that happens in the dungeon. • Report the IP address for your server. • Use radio buttons to turn the actual serving for clients (the listening) on and off.	**25**	**Yes**	In the case of the requirement about radio buttons toggling the server “listening” to clients, I will take “listening” to mean that a client’s requests are met by a server response. In other words, if the radio button is selected, the client requesting to move in a certain direction or pick up a piece of gold will be handled by the server. If the radio button is de-selected (the server is NOT listening), the request will be ignored.

Create a GUI which launches the server on any chosen port number.	• Allows a user to choose a port for the server to run on, before launching that sever on the designated port at the click of a button. • Allow a new port to be selected, but disable this if the server is currently serving / listening	N/A	Yes	**This GUI must launch a server first before a client can be launched to prevent connection errors being thrown due to the client having no valid server to connect to.**
Create a GUI which launches a client (human or bot) and connects it to the server on a chosen IP address and port number.	• Allows a user to choose an IP address and port number combination for the client to connect to a running server on, before launching that player (human or bot) by opening their player GUI at the click of a button	N/A	Yes	**This GUI must launch a client after a server has been launched to prevent connection errors being thrown due to the client having no valid server to connect to.** This simply means **"ensure the port number and IP address entered matches a running server before launching a client".**