CM10228: 'Dungeon of Dooom' - Coursework 2: Adding a GUI (Java): Documentation of Program

Program Documentation/Instructions

The Dungeon of Dooom is played on a rectangular grid (the Dungeon) on which the player can move and pick up items. The goal is to collect enough gold and make it to the exit.

You start the game with no gold, and at a random location within the dungeon. This position may contain gold (if you are lucky), may be an empty tile, or it may be an exit tile.

The objective of the game is to collect at least a certain amount of gold and then move onto an exit tile in the dungeon. If you have enough gold and land on the exit, you automatically leave the dungeon and the game finishes.

Running the game in eclipse will give optimum performance. On LCPU the performance is slightly hindered.

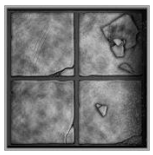
Tiles

The dungeon is made out of square tiles. A tile can be:

Look Window:

Floor: Allows a player to walk over it, some may also contain gold.

Empty tile:



Gold tile:

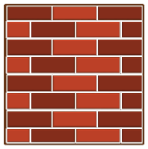
Player:

Current player (Human):

Current player (Bot):

Other player:

Wall: Prevents a player from moving through it.

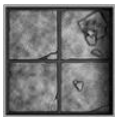


Exit: A special floor tile necessary for winning the game.

X tile: A tile that is out of view of the current player. This is a tile beyond a radius of 2 units away.

God's Eye View:

Empty tile:



Player:

Gold tile:

Wall:

Exit:

Commands

“MOVE <direction>”

Moves 1 square in the indicated direction.

Response: “SUCCESS” or “FAIL”

“PICKUP”

To pick up the item in the player’s current location. On success, the new total of gold left before a win is displayed in the client window and their progress bar updates.

“ID”

Returns the Client ID number of the player requesting their ID. Displays this ID in the text area.

“QUIT”

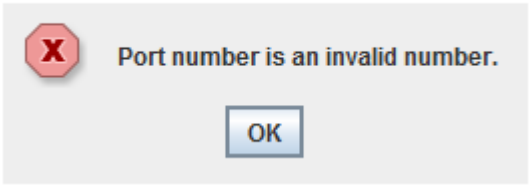
Exits the game client-side and disconnects the client from the server. Other players are left connected whilst they continue to play.

NB: There may be no way to win the game if your bot has collected all the gold.

How to Play

1. **Run StartServer.java**. This will open the GUI responsible for choosing a port for the server to run on. Choose a port number between 1 and 65,536. Type this number into the editable text field and click 'Start Server' to launch the server on that port.

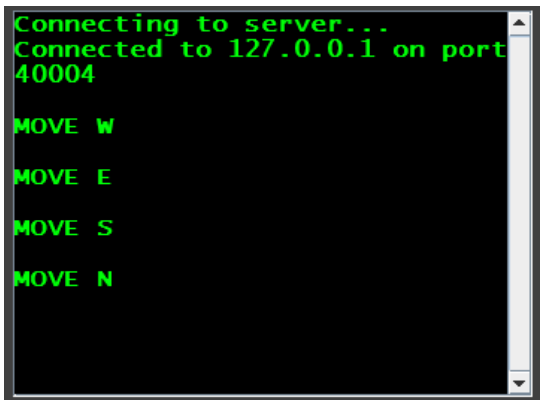
If the port number is invalid (outside the range of 1 < n < 65,536), an error message will appear.



2. **The GameEngineGUI will load.** The port number and IP address of the server are reported. The map is loaded and GUI constructed, with radio button and control button for listening to clients and shutting down the server respectively. **The GameEngineGUI shows the whole map, which is different from the look window (which is a restricted 5x5 grid view of the portion of the map around a given player).**

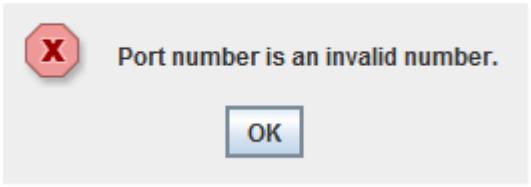
The radio button may be selected and de-selected to turn the actual serving ("listening") to clients on and off.

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.

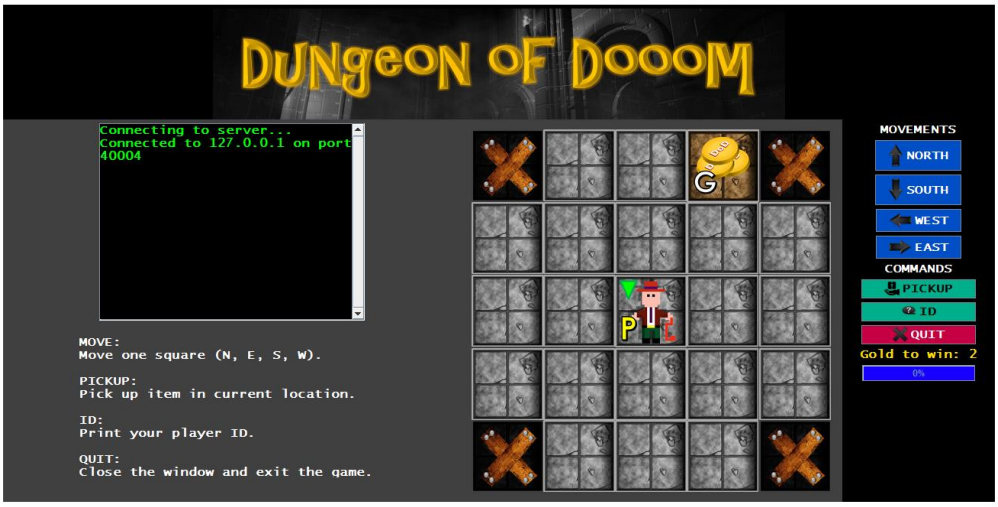


3. **Run StartClient.java**. This will open the GUI responsible for choosing a port number and IP address for the client to attempt to connect to. Choose a port number between 1 and 65,536. Type this number into the editable text field. Type either "localhost" or an IP address of the form xxx.xxx.xxx.xxx into the IP address field.

If the port number is invalid (outside the range of 1 < n < 65,536), an error message will appear.



4. **Click 'Launch Human'** to launch a human player into the game. The human player UI contains control buttons to move around the dungeon and pick up gold. It also allows the user to check their ID. Finally, the user may quit the game by clicking "QUIT". **A player can only see a 5x5 grid of what objects are situated around it. No player has access to the entire map.**



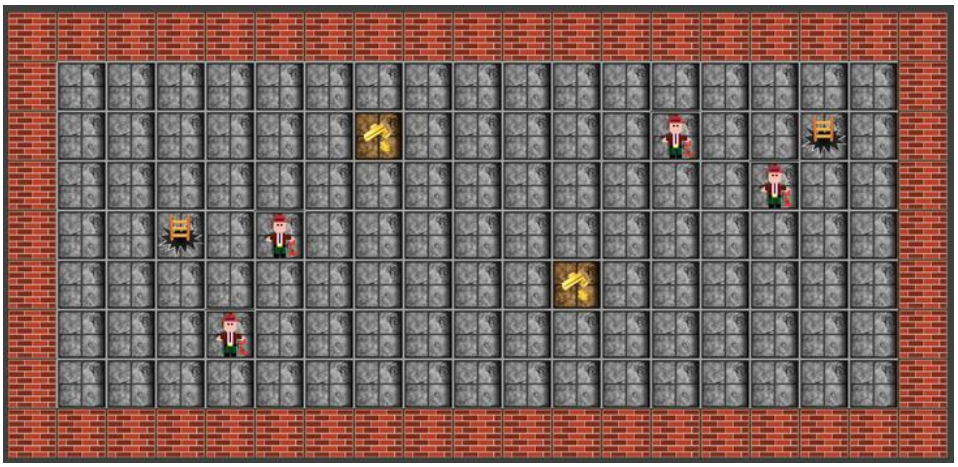
5. **Or click 'Launch Bot'** to launch a bot player into the game. The bot UI contains no navigation buttons, as the bot moves automatically in the dungeon. The bot will automatically pick up any gold that it lands on. **A bot can only see a 5x5 grid of what objects are situated around it. No player has access to the entire map.**

6. **If the player is on top of a gold tile**, clicking "PICKUP" if you are a human player will return "SUCCESS" and the "gold to win" tally will decrease as that player needs one less piece of gold to escape the dungeon. Also, the progress bar will reflect the progress made.

NB: Bots pick up all gold that they land on automatically, hence their player GUI doesn't include a pickup button.

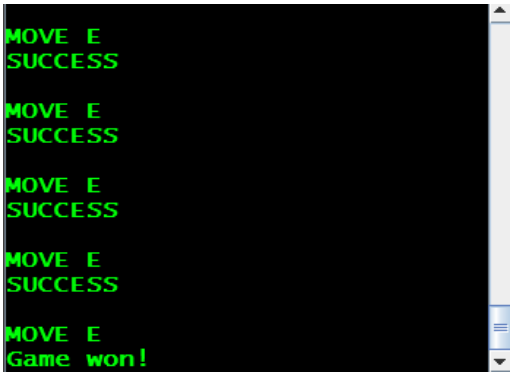
7. **If another player is present in the dungeon around you**, you will see them show up in red in the player look window.

NB: On the God's Eye View, all players appear the same.



8. **The aim of the game is to collect enough gold to win the game and reach an exit point before any other player**. When enough gold has been collected, the gold required to win shows up as 0, and the progress bar for that client displays as 100%.

The goal at this stage is to reach an exit before any other player. If a player manages this, they have won. A message is displayed to the user if this happens and the **server closes**. As a result, command buttons are frozen and the user must click "X" in the top right corner to exit cleanly.



Critical Analysis

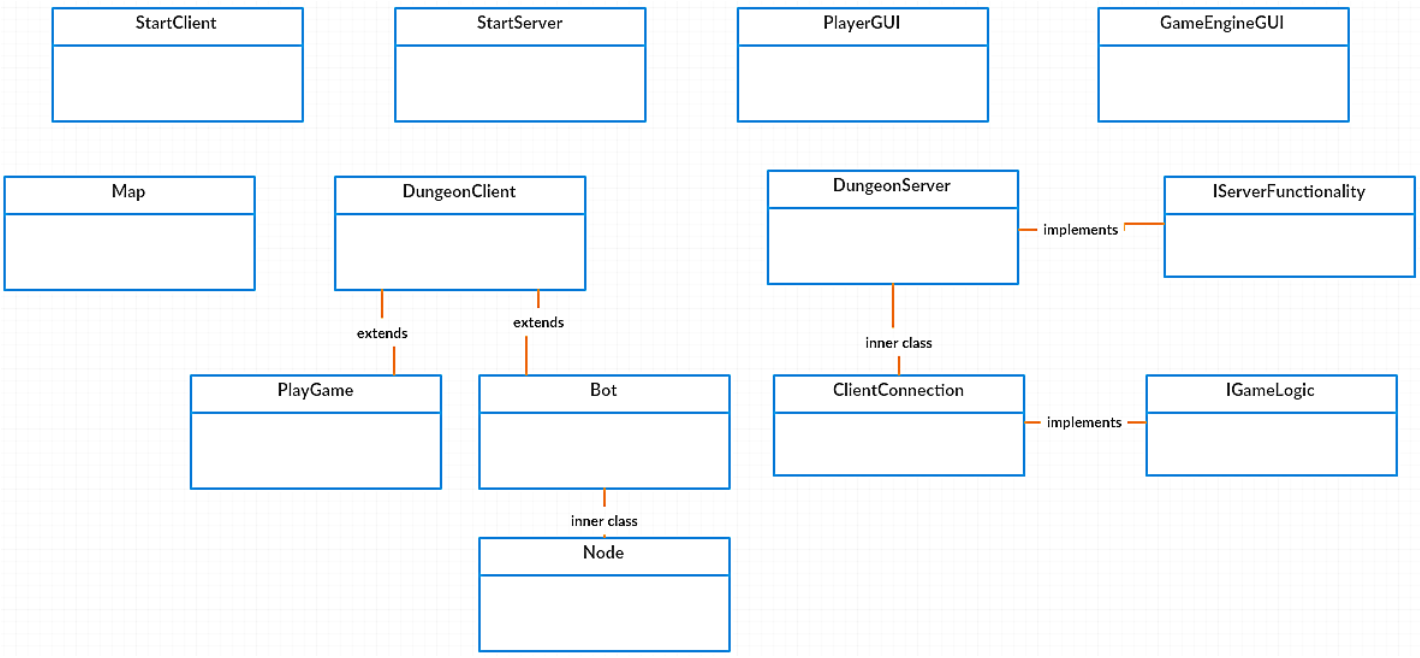
What did you do right?

I met all of the requirements of the basic GUI. My PlayerGUI opens up in its own window and contains a control panel of buttons allowing a player to move around and pick up gold in the dungeon. It also shows the user the outcome of their actions by containing a text area printing server feedback. I therefore met the basic requirements of the GUI set out in the brief.

I built 4 GUIs in total. StartServer is responsible for launching the server on any chosen port number. Once a valid port number is entered and the user clicks “Start Server”, the GameEngineGUI launches. This loads the GUI responsible for displaying the God’s eye view of the map with players on top, as well as a radio button to turn listening to clients on and off, and a control button to shut the server down.

Once a server is up and running, running the StartClient GUI allows a user to choose a port number and IP address that gives the user a choice of which server to launch the human or bot player into. Upon entering valid credentials in these fields and clicking “Launch Human”, a human player is launched into the dungeon with command buttons to move around and a graphic pane to display the look window of the player. The same is true for bot players, however moves are made automatically and there are therefore no movement command buttons. Both types of player have quit buttons, a text pane to

display the result of actions and a progress bar to track how close the player is from escaping the dungeon.



All the basic functionality and game logic works for both types of player. **I chose to work on advanced feature 1 and advanced feature 3.** I feel I completed these well. The graphic pane is effective and updates not only when a client moves but also on a regular timed basis (every 500ms). This means that each player's graphic pane updates even when they are standing still. The pane updates for each player move and each bot move.

Progress through the game is shown by recording moves made in the text pane next to the graphical window, as well as including a "gold to win" tally and progress bar to accompany this. It is important to note that **this tally is player-specific** – in other words it **shows the number of gold pieces that an individual player needs to collect before they can escape** the dungeon.

The GameEngineGUI works well. It contains a "God's eye view" panel 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. The IP address of the server is reported when the server starts. If the radio button is selected, client commands are met with a response. If it is de-selected then the request made by a client is ignored by the server. The port of a server can be changed using StartServer.java, however this will not work if a game is currently running. This is because the current game is run using a server socket connection, which can't simply change its port without closing the socket and disconnecting all connected clients.

What did you do wrong?

There is ambiguity in advanced feature 3 over what "listening to clients" actually entails. In this case, I assumed "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.

As such, I made sure that the radio button I created performs to this purpose. New clients can still connect when the button is de-selected, but their requests to move and pickup gold are not listened to.

However, I am also aware that "listening" can be taken to mean "does it accept new clients". I believe I completed the advanced feature well, but I would have made sure next time that I obtained clarification as to what the requirements means before completing it.

I also experienced **issues with my Bot AI** during the development of this project, meaning I had to revert the movement strategy from using the Breadth-First algorithm back to moving randomly. Seeing as the Bot AI doesn't come under the assessed features I have chosen to implement, it doesn't carry any marks. This means that the AI reverted back to being simple for the sake of minimising errors in the program and ensuring the GUIs (which do carry marks in this coursework) function to their intended purpose.

Next steps (possible improvements)

What would you do differently if you did it over?

If I was to complete the project from scratch again, I would redesign my structure such that the client/server architecture works on a 'server push' basis instead of a 'client pull' behaviour. I would have Player GUI display updates in response to events from the server. That is, let the server send an event to the Player GUI and have that GUI update even though the player hasn't done anything. For example, an AI bot may have walked into the room. This would reduce the number of requests the client needs to make of the server and just means that if there are any changes, the server simply has to push all of these changes out to all clients at once. This is a more efficient way of organising the retrieval and update of data and ensuring clients have an updated copy of their look window.

I would also improve the AI of the bot player so that it finds gold and retrieves gold a lot quicker than it did. Perhaps it would be a good idea to choose the speed at which the bot moves to ensure there are different difficulty levels. Further, different bots could have different path finding algorithms to ensure they are more/less effective than other types.

Once a player has won, it might be a nice idea to total the number of moves they made in order to win the game. These scores could be stored somewhere along with usernames and used as a leader board to compete with friends and family.

In terms of graphical improvements, it might be good to have different 'texture packs' so that the user can choose a character and dungeon skin which improves the variety of styles of gameplay and introduces a kind of customisation to the game. I might want to include more messages to the user which presents feedback – for example if they attempt to move into a tile where another player is standing, I might want to print a message such as "Oops! Player <ID> is standing there!".

Further to the ambiguity over what the "listening" entails, I might want to extend the "not listening" code to ensure the server stops listening for new connections as well.

In terms of code structure, it would be good to stick by the MVC Model Viewer Controller model a little better and ensure that the common elements of each GUI - control buttons, layouts etc. all sit in their own class instead of encapsulating each new GUI in its own class. This would mean as the GUI grows in size and complexity, the amount and organisation of code doesn't become unmanageable.

I would like to learn how to implement keyboard commands – this would mean that using the arrow keys on the keyboard moves the player north/south/west/east and the P key could mean "PICKUP". This would make a nice extension to the way commands are executed and would allow me to do research into key listeners etc.

I could extend the protocol by adding more possible items into the dungeon (e.g. weapons, cameras, mobile phones). Even though the PlayerGUI does change if a bot is playing, I might want to make another GUI for launching AI bots, or add a pane to the Game Engine GUI to do this.

A chat capacity could be added to the PlayerGUI which would mean connected clients can send messages through the server to other clients. This could allow communication with associated usernames.

Given more time, I would attempt to work on at least a few of these features to improve my program.

See below for screenshots of my GUIs

Screenshot 1 – StartServer.java

Screenshot 2 – StartServer.java

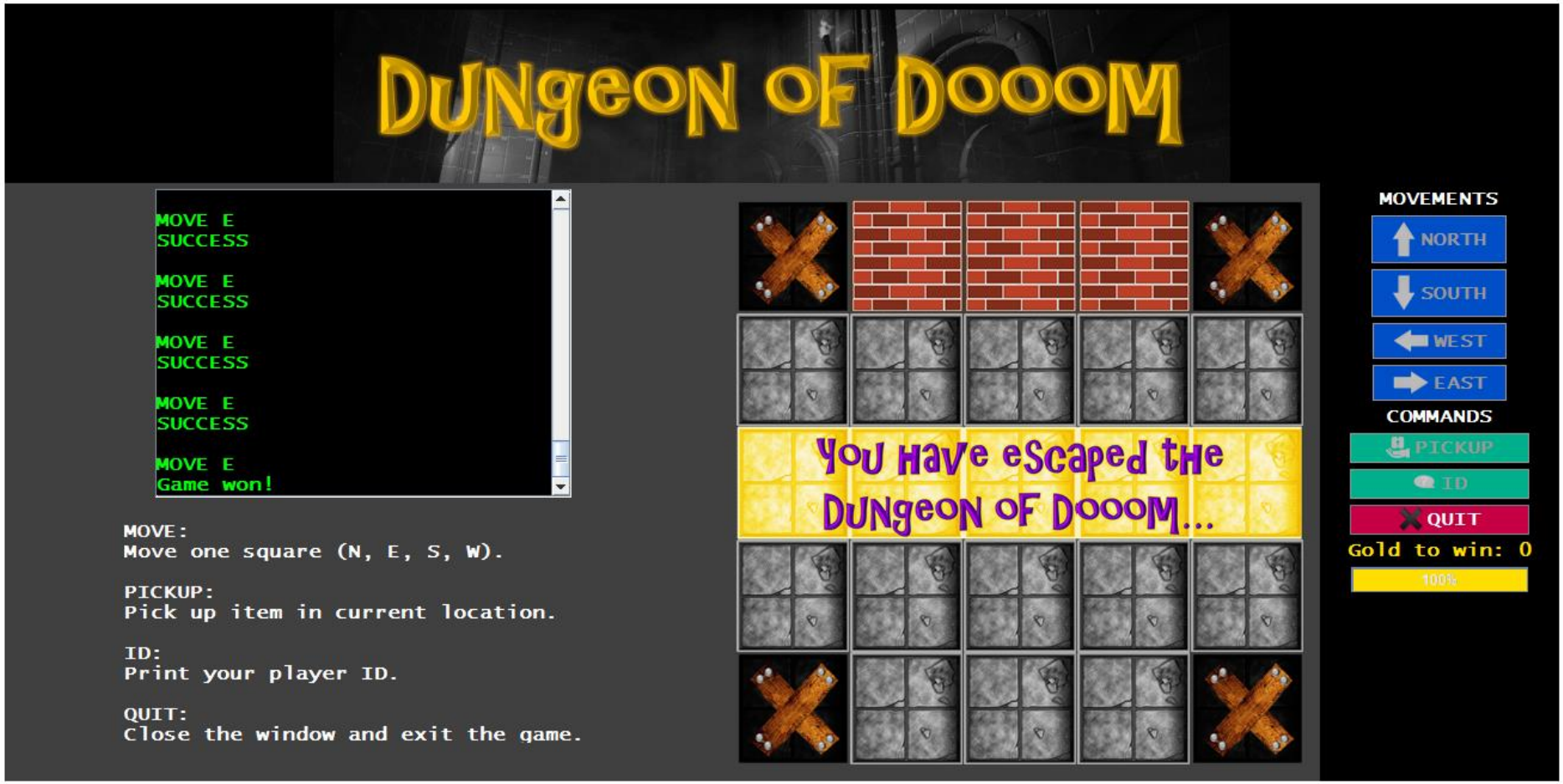
Screenshot 3 – GameEngineGUI.java

Screenshot 4 – StartClient.java

Screenshot 5 – PlayerGUI.java (Human mode)

Screenshot 6 – PlayerGUI.java (Bot mode)

Screenshot 7 – PlayerGUI.java (winning a game)



Screenshot 8 – GameEngineGUI.java (not listening to clients)

Screenshot 9 – PlayerGUI.java (Human mode - not listened to by server)

Screenshot 10 – PlayerGUI.java (Human mode – server socket closed)