Dungeon Of Doom Coursework 3

Documentation

Running the program

First, you must compile the files for Dungeon of Doom, so that it understands your computer. First, navigate to the DungeonOfDoom directory, inside the provided archive. Then, open your terminal application and run the command:

```
javac -d bin -cp src src/*/*.java
```

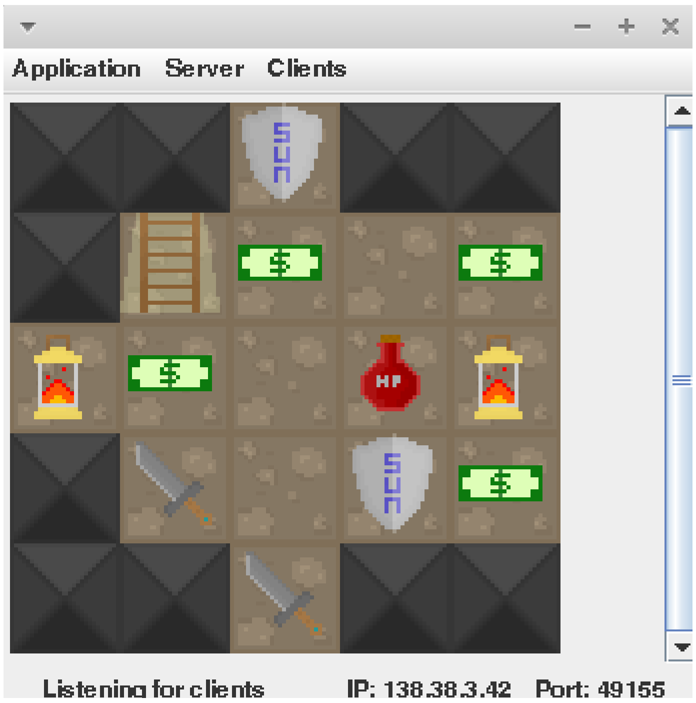
When the command has finished, you may launch different parts of the application with the following commands:

A visual client for playing the game:
```
java -cp bin dodClients/GUIGameClient
```
A command line client for playing the game:
```
java -cp bin dodClients/CLUIGameClient
```
A visual application for managing a server:
```
java -cp bin dodServer/GUIServer
```
A command line client for running the server:
```
java -cp bin dodServer/CLUIServer
```

The server applications will ask for a path to a map file. To use the default, just enter 'defaultMap' here. They will also ask you for a port. Pick a number you like, ideally between 49152-65535.

The client will ask for your port from earlier, and also the IP address. This will be displayed on the GUI, or run the 'ip' command for the CLUI server. Using the command line client, you must use the commands detailed in the Dungeon of Doom specification.

Using the program



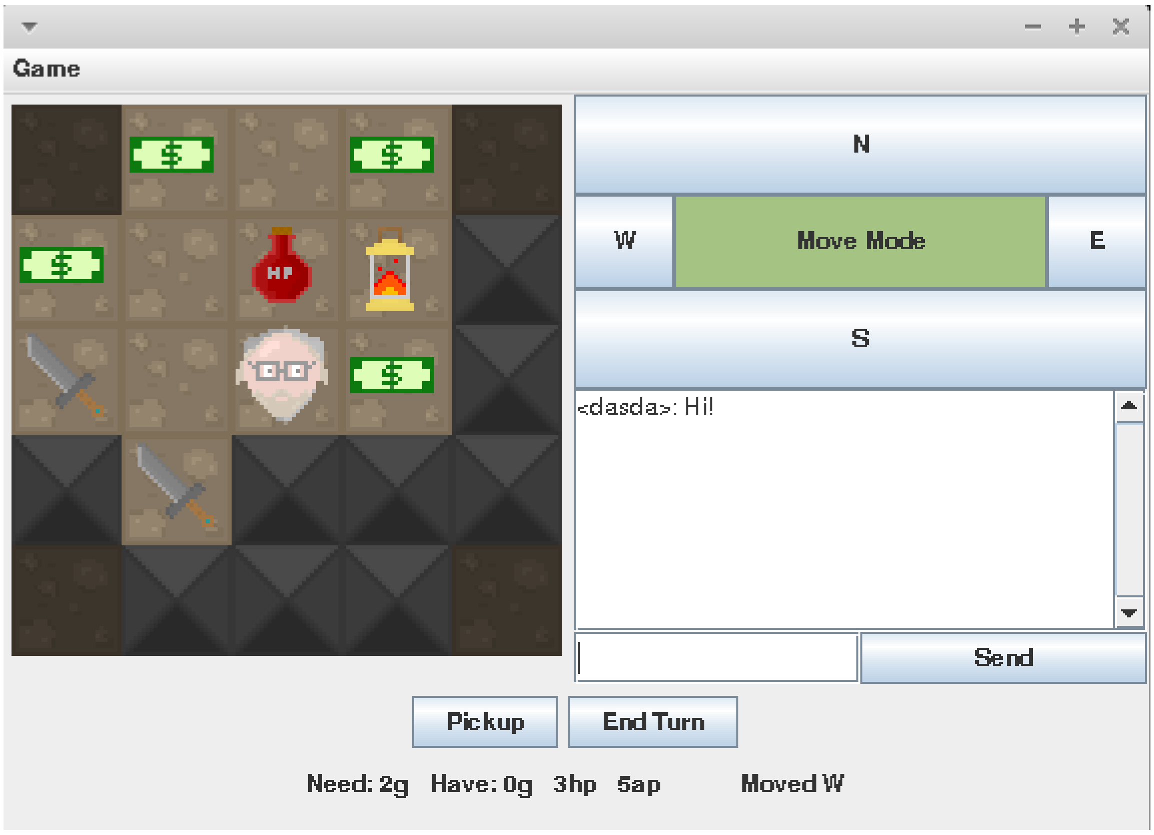
Listening for clients IP: 138.38.3.42 Port: 49155

Here can be seen the Server GUI. When the server GUI is run, you will be prompted for a map name and port. Enter them, and this screen will appear.

As players move around the map, it will update. If you are playing too (and don't want to cheat!), the map can be turned off by clicking Application → Hide Map

Serving can be turned off using the buttons under the 'Server' menu. Finally, when the server is stopped, you may change the port by clicking Server → Change Port.

If you want to make the game more challenging, you can add a bot to the game by clicking Clients → Add Bot



<dasda>: Hi!

Need: 2g Have: 0g 3hp 5ap Moved W

Run the GUI client, enter the port and IP, and you'll see this screen. Clicking the 'Move Mode' button will change the N, E, S and W buttons between move and attack. Hitting 'Pickup' will let you pick up items. They do all sorts of beneficial things, so grab as many as you can. 'End Turn' will end your turn immediately, so watch out. You don't want to waste moves! Type in the text box and click 'Send' to talk to/taunt the other players in the game.

Analysis

On the whole I managed to implement all the intended features for this coursework. However, it was harder than I initially anticipated, with me having to make rather significantly more extensive changes to the GameLogic and associated classes, as well as the background functionality of the server.

I made the Client GUI first, and this worked out well because I was able to reuse quite a significant amount of the assets and classes from the Client GUI when creating the Server GUI. I was also able to apply the knowledge learnt about Swing in order to create the Server GUI much faster.

When creating the Client GUI and Server GUI, I did not consider extensibility very much, or that those Jframes might need to be accessed separately without a main method. When closing dialog boxes and the Client and Server GUI frames, these classes call System.exit. This was a mistake, as I later discovered when I wanted to allow the Server frame to launch an instance of the Client frame and automatically connect it to the server. Clicking the close button on that frame would take the server down with it. The way exceptions are handled in some of the constructors for these classes also prevent these GUIs from being launched without the dialog boxes appearing, at least without some rewriting. As a result, this feature had to be abandoned.

Despite this, I feel that the structure of my code has improved since coursework 1, with significant improvements made simply by looking at the code base given for coursework 2. This has taught me that a lot can be learnt from reading other peoples code, and I will be making an effort to do this more in the future. I am particularly proud of some of the NetworkMessageListener interface, as it made making a variety of clients trivial.

Additional to the features listed in the features table, I also went ahead and implemented a few extra features I felt that would make this program a better piece of work overall. I implemented a basic chat client, as well as a combat between players. I made a number of changes to the behaviour of the game – for example, players, when removed from the map, drop all the gold they are carrying (within reason) on the tile beneath them. They are then moved out of bounds of the map, so they can continue interacting with other players in the game, without affecting the game. They also don't get turns in this state

As can be seen from the screenshots, the GUI does not match the designs very well. There were two reasons for this. Firstly, I discovered that implementing the kind of GUI shown for the client is harder than it looks as it isn't really provided by the features of the Swing layout managers. Secondly, I decided menus were more attractive and easier to use for the server GUI, where many features are provided.

If I were to start this program from scratch, I would definitely pay greater attention to the extensibility of the classes as I coded them. I would also ensure that the specification was more detailed than it was here. Increasingly I have learnt that planning is key when starting programming projects. The plan here was too rough, and didn't consider enough details sufficiently.

Specification

Feature Table

The features I am aiming for are as follows:

Intended Features	Points	Self Evaluation	Marks & Checker Evaluation	Special Instructions
Basic Client GUI	15	Yes. All features implemented, and unable to find any bugs from my testing.		None
Client Graphic Pane	10	Yes. Provides a visual way to display the map, and extensible for larger look replies. No bugs found.		None
Updating graphic Pane	10	Yes. Implemented the 'CHANGE' feature in the wire spec. Any changes to the map, immediately cause a 'LOOK' to be sent and the map updated.		None
Server Game Engine GUI	10	All the features on the spec implemented, and a few more. No bugs. The graphic pane for this GUI also updates. Would like to be able to kick players and bots, but didn't have time.		None

Class Description

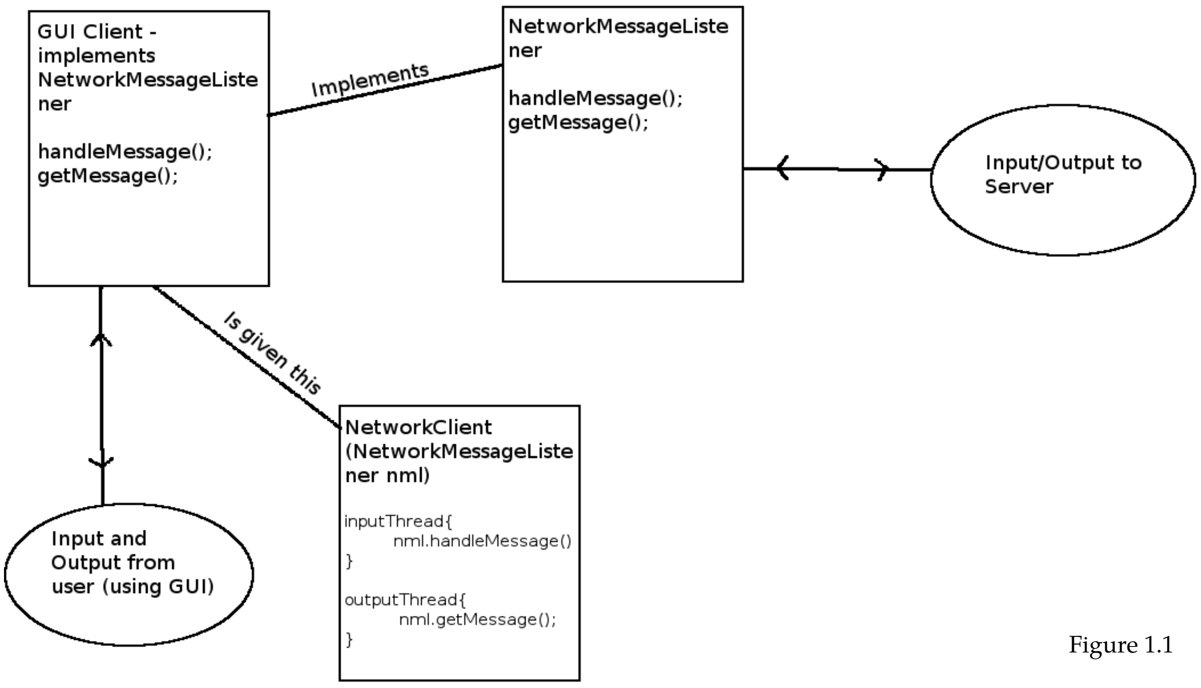

GUI Client - implements NetworkMessageListener
handleMessage();
getMessage();
Implements
NetworkMessageListener
handleMessage();
getMessage();
Input/Output to Server
Is given this
NetworkClient (NetworkMessageListener nml)
inputThread{ nml.handleMessage() }
outputThread{ nml.getMessage(); }
Input and Output from user (using GUI)

Figure 1.1

A diagram of the design of the Client GUI can be seen in figure 1.1. To implement the Client GUI, I will use the existing Dungeon of Doom client – the NetworkClient class, which takes an interface called the NetworkMessageListener as an argument in its constructor. This interface provides two methods, one which gets a command to send to the server (getMessage()) and one (handleMessage()) which gives a command from the server to the implementing class. The GUI class will simply implement this interface, and have buttons give commands to the server by putting them on a stack for the getMessage() method. When the handleMessage() method is called, the UI will update accordingly.

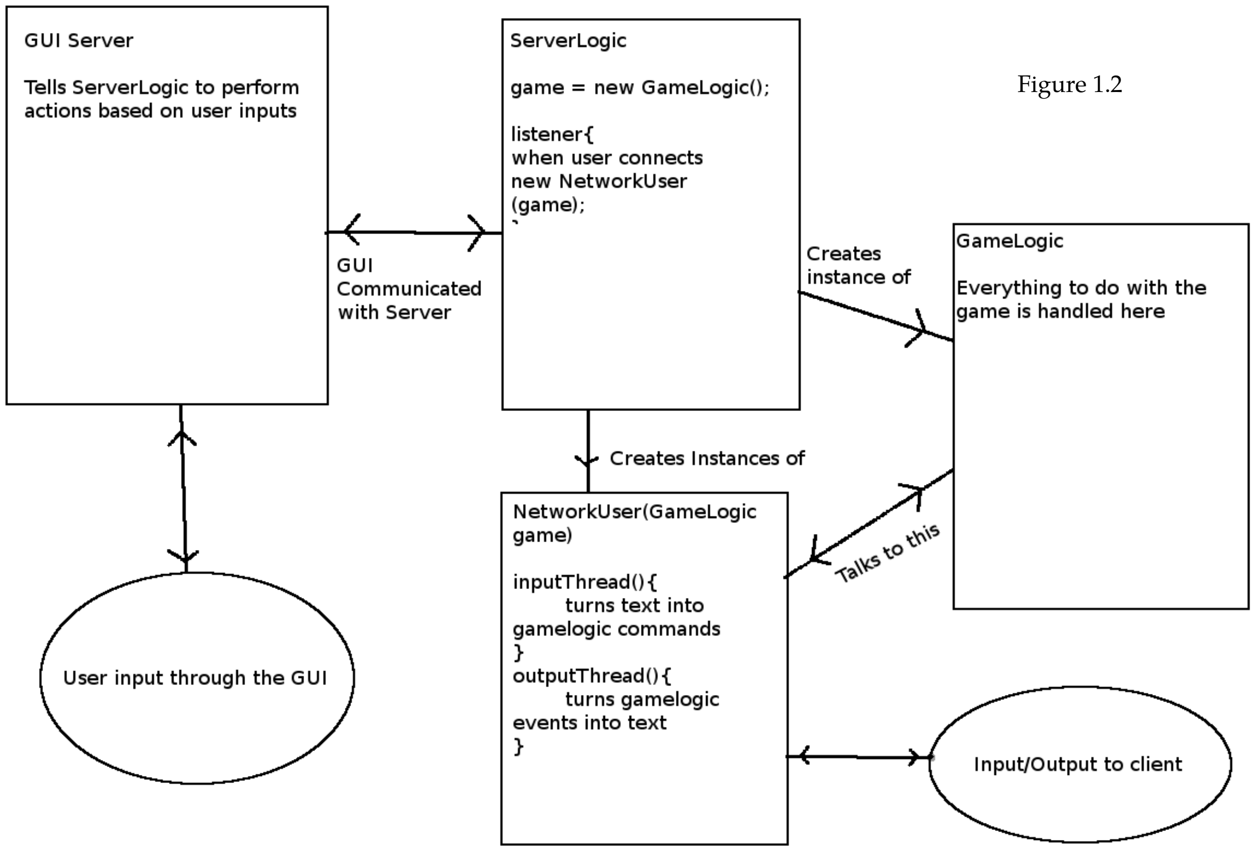
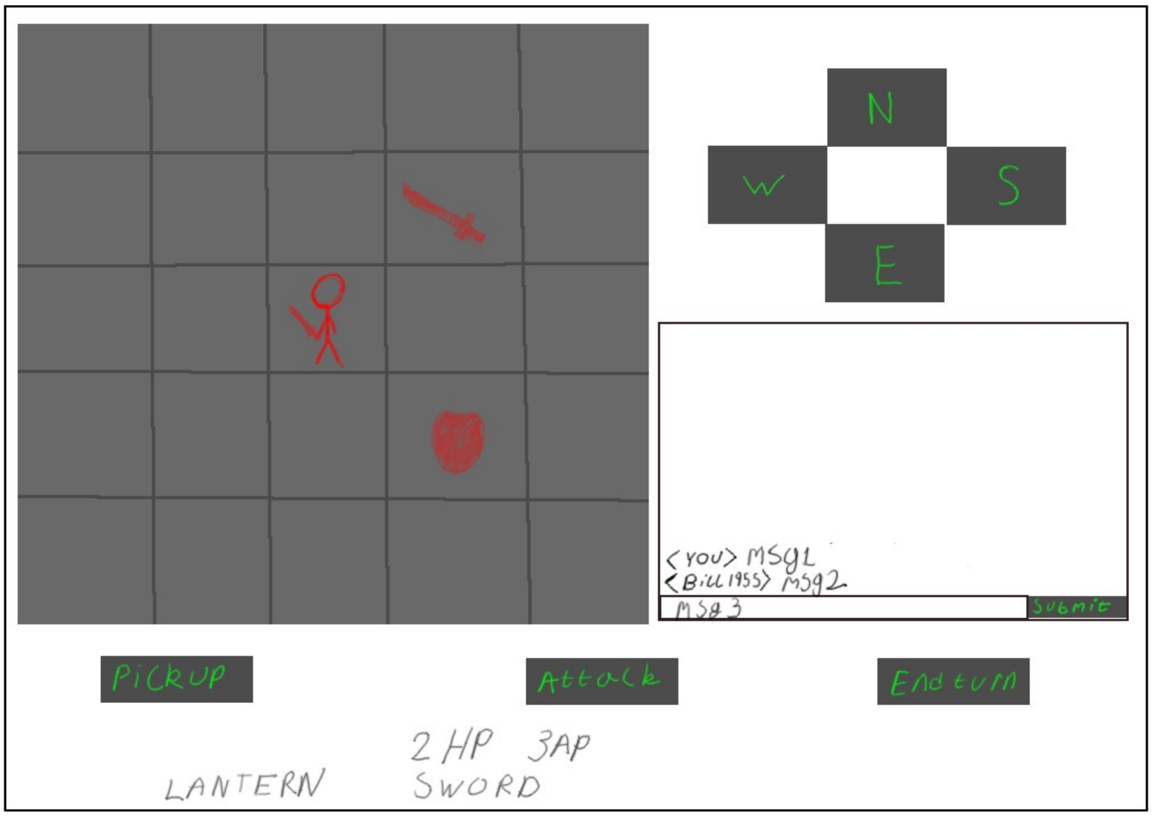

GUI Server
Tells ServerLogic to perform actions based on user inputs
ServerLogic
game = new GameLogic();
listener{
when user connects
new NetworkUser
(game);
GUI Communicated with Server
Creates instance of
GameLogic
Everything to do with the game is handled here
Creates Instances of
NetworkUser(GameLogic game)
inputThread(){
turns text into gamelogic commands
}
outputThread(){
turns gamelogic events into text
}
Talks to this
User input through the GUI
Input/Output to client

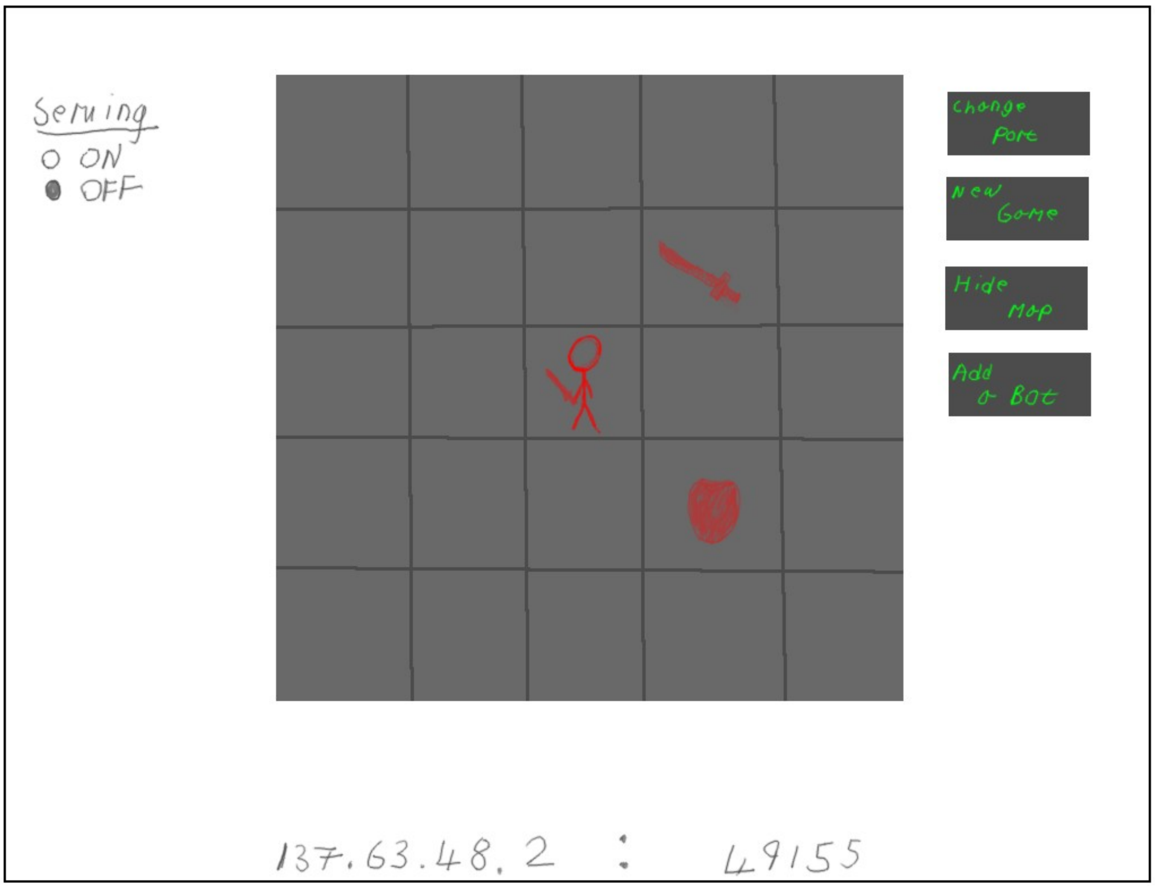
Figure 1.2

A diagram of the Server GUI design can be seen in figure 1.2. I will take the existing Server class and remove all of the input and output functionality from it. I will then add methods to this new ServerLogic class which provide the ability to disable listening, enable listening, change ports and get a character array representation of the map from GameLogic. The GUI class will simply call methods from ServerLogic to perform actions.

The ServerLogic class will launch a listener thread that waits for a new client to connect. When it does, that client will be handed off to an instance of 'NetworkUser', that is given the GameLogic instance. This way, the client can talk to the game part of the server (using the commands specified in the wire spec)

I have drawn diagrams of both GUIs – the client GUI and the server GUI. These diagrams can be seen on the following page. These diagrams are fairly self explanatory, buttons simply represent the actions that they will perform. Clicking the 'Attack' button on the client GUI will switch the 'N, E, S, W' buttons to attack in the matching direction. Clicking it again will toggle them back to move. The buttons will change colour to indicate this.


N
W
S
E
<you> MSg1
<Bill1955> MSg2
MSg3
Submit
PickUP
Attack
End turn
2 HP 3AP
LANTERN
SWORD


Serving
○ ON
● OFF
Change Port
New Game
Hide Map
Add a Bot
137.63.48.2 : 49155

Test Plan

Test Number	Test Case	Test Data	Expected Output
1.	(Client GUI) Character moves in the correct direction when the appropriate button is pressed.	Click the 'N' button. Repeat for 'E', 'S' and 'W'.	The next LOOKREPLY indicates the player having moved the specified direction.
2.	(Client GUI) Character picks up item beneath them when the 'Pickup' button is pressed.	Click the 'Pickup' button over a Lantern, then over an empty square.	The next LOOKREPLY will indicate a lantern has been picked up. 'Lantern' will appear at the bottom of the screen. On an empty square, nothing should happen.
3.	(Client GUI) the message in the message box is sent to all connected users when the 'Submit' button is pressed.	Two clients should be connected. A message should be typed into the message box, the submit pressed. Repeat with empty box.	The message should be relayed to both clients, with the players name as a prefix. If the box is empty, nothing should happen.
4.	(Client GUI) If the End Turn button is pressed on the players turn, the players turn should end	When it is the players turn, click end turn.	The turn should be passed to the next connected player.
5.	The player should be able to attack other players.	With another player next to the player, press attack in each direction. Repeat with other player in a different direction.	The other player should take damage, if there is a player there.
6.	(Client GUI) The move, pickup, end turn and attack buttons should not work on another players turn.	When not the players turn, try each of the move, attack, end turn and pickup buttons.	Nothing should happen – the user should be informed that it is not their turn.
7.	(Client GUI) When another player moves, the visual display should update.	Connect a second client, and use the move buttons to move it around.	The first client should see the second client's player moving around.
8.	(Server GUI) The Serving radio buttons should turn serving on and off.	Turn serving off, then attempt to connect a client. Then turn it back on, and try again.	The client should fail to connect. After turned back on, the client will connect.
9.	(Server GUI) The New Game button should start a new game.	Click the new game button.	A new game should start.
10.	(Server GUI) The Change port button should change the port.	Start on port 49155. Click the new port button. Enter the new port (49156). Connect a client at 49156.	New client should connect.
11.	(Server GUI) Clicking hide map should hide the map.	Click hide map.	The map should no longer be visible.
12.	(Server GUI) Clicking add bot should add a bot.	Click 'Add bot'	A new bot will be added to the game, and will appear on the display.