# Final GUI

# 

# Playing The Game

# Automatic Configuration

The included script will compile and run the game for you.

To use the included script:

1. Open a terminal and navigate to the root of the game directory (DungeonsOfDooom)
2. Run the Play.sh script using the following command (without the quotation marks): “./Play.sh”
3. This will compile and start a server and client in GUI mode
4. If you would like a different configuration, use the manual configuration option below

# Manual Configuration

## Compilation

1. Open a terminal and navigate to the root of the game directory (DungeonsOfDooom)
2. Type the following commands (without quotation marks):
3. “mkdir out”
4. “cd src”
5. “javac -d ../out \*.java”
6. cd ..

## Running the game

### Running the server

#### GUI Mode

1. Open a terminal and navigate to the root of the game directory (DungeonsOfDooom)
2. Enter the following command: “java -cp out Server”
3. If you would like to specify a port, type the port number at the end of the command, leaving a space before
4. A file choosing dialog will open, allowing you to select a map

Example: “java -cp out Server 40004”

In this case, 40004 is the port number of the server.

#### Command Line Mode

You can also run the server in command line mode. Follow the instructions for GUI mode, but add a ‘nogui’ argument to the end of the command.

Example: “java -cp out Server 40004 nogui”

In this case, 40004 is the port number of the server.

### Running the client

#### GUI Mode

1. Open a terminal and navigate to the root of the game directory (DungeonsOfDooom)
2. Enter the following command: “java -cp out PlayGame”

#### Textual Mode

As with the server, you can also run the client in command line mode. Append the command for GUI Mode with a ‘nogui’ argument. You will also need to specify the ip address and port number of the server as arguments.

Example: “java -cp out PlayGame nogui 127.0.0.1 40004”

In this case, 127.0.0.1 is the IP Address of the server, and 40004 is the port number.

# Analysis

## What I did right

### Fog of War

The client only receives its visible look window when it looks, and keeps track of it position relative to where it spawned. Using these two features along with the ClientMap class, it builds ang-of-war style expandable map as it moves around. This allows for a complete view of discovered areas, whilst preventing cheating and reducing the size of look packets. Other players are only displayed in the current look window around the player, regardless of where they have previously been shown.

### AI

My AI uses a complex task based system of my design. The bot class contains a stack of tasks, and gets commands from the top task before removing it from the stack once it is complete. The base task (ExploreTask) is only complete once the game is won.

I have implemented A\* pathfinding in the ClientMap class, which the bot uses to remember where it has been. This allows the ExploreTask to explore efficiently, by finding the nearest reachable, unexplored area and getting the fastest route to that area. Pathfinding is adaptable, so if a player gets in the way, the bot will move around them. I have tested this in complex mazes, and the bot consistently behaves in an intelligent way.

The explore task does this exploration activity until it finds a piece of gold, at which point it adds a CollectGoldTask to the stack. If it doesn’t need gold, it will move to the nearest exit instead.

### MapPanel

My map panel has continuous, smooth transitioning between tiles, even though the player position is quantized to the nearest tile. This gives the appearance of smooth walking.

The player is also rendered facing the direction they last moved, adding polish to the game.

## What I did poorly

### Asynchronous Client

My client runs two threads asynchronously. The InputListenerThread takes input from the PlayerInterface and sends it to the server. The ResponseListenerThread takes output from the server and gives it to the PlayerInterface. I did this in an asynchronous way to allow the server to update the client unexpectedly, but it causes issues with matching responses to their relevant commands. I had to add additional workarounds to make sure that the response is always matched to the correct command, such as only allowing a new command of any type once the response has been collected from the last command call of that type.

#### How I Would Improve it

I would still keep the sending and receiving in different threads, but would include synchronisation in certain methods to avoid issues. This removes the need to sleep for 20ms in each thread as well, improving responsiveness.

I would also give each packet an ID, and wait for the response to that packet before allowing a new command to be sent.

I would also remove the process of the client requesting look windows, and simply have the server send them whenever something changes. This reduces network traffic.

### Player Data

The original design of the game was restricted to a very strict protocol. This forced me to include other player positions in the map look window response from the server. This is bad for a number of reasons:

* The map data underneath the player is lost
* The direction of the player and other relevant data cannot be sent
* When building a fog of war style map, other players can sometimes be duplicated, which requires workarounds

#### How I Would Improve It

Player data would be sent separately to map data. This would allow me to send player directions, actions, animations, and positions as well as complete map data inside the look window. This would allow for more expansion of features and reduce bugs and workarounds.