



Get Rich or Die Trying

User Manual

Version 1.0
15/12/17

Contributors:
Cheuk Hei Chan
Kimberley Chong
Alex Gooding
David Hurst
Jessica Lok
Megan Slattery

Table of Contents

INDEX	PAGE
1 What is Get Rich or Die Trying?	
1.1 Introduction	1
1.2 Objective	1
2 Running the Game	
2.1 Before running the game	1
2.2 Controls	1
2.3 Starting the game	1
3 Playing the Game	
3.1 User interface	2
3.2 Winning the game	3
3.3 Dying in the game	3
3.4 Leaderboard	3
4 Quitting the Game	4
4.1 Main menu	4
4.2 In game	4
5 Contact	4

1 What is Get Rich or Die Trying?

1.1 Introduction

Fresh outta' cash? Runnin' on empty? On a quest for quid? Welcome to Get Rich or Die Trying: the rogue-like game where you can find your fortune. Roam through the mansion of money and collect as many gold coins as possible. But careful. This mansion might deliver in dreams, but death is also possible. Ghosts lurk in every room, guarding the precious gold. Touch one—you die! Avoid them, grab the gold, unlock the escape portal—you're made! Do you dare to get rich or die trying?

1.2 Objective

The objective of Get Rich or Die Trying is to make your fortune by collecting as many gold coins as possible. Move through all 3 floors in the mansion of money, grabbing 3 coins on each level. After this, your escape route appears. But remember—there's no such thing as free money, as this game has its price. In each room resides a ghost! Every time you move, all the ghosts in the map move and touching one means instant death. Too easy? Then try to reach gold, silver or bronze on the leaderboard. The top-spots can be obtained by completing the game in the fewest number of steps. Good luck with getting rich, we hope you don't die trying.

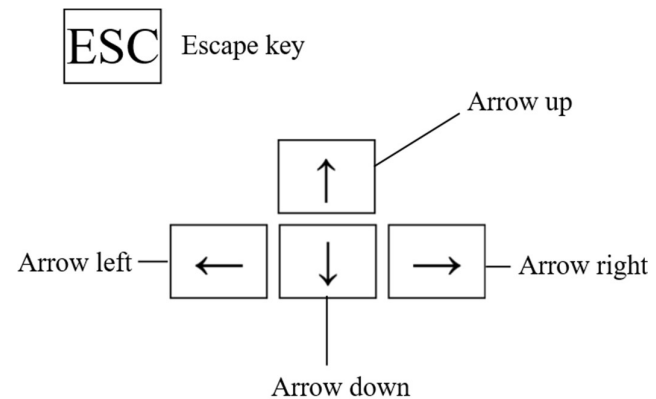
2 Running the Game

2.1 Before running the game

Be cautious before running the game as the volume may be quite high at the start of the game. Listening at a high volume for a long time may damage your hearing. Tune the volume of your system to safe levels to reduce impact.

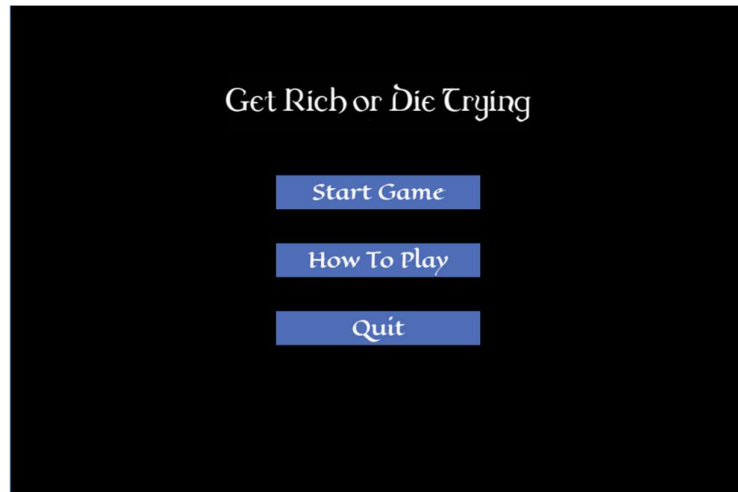
2.2 Controls

Your primary game input device is your keyboard. Up (↑), down (↓), left (←), or right (→) arrows moves the player in the expected direction. The Escape key, exits the game. A pointer (mouse or trackpad) is also necessary for interaction in the game menu.



2.3 Starting the game

Ensure that you have followed the instructions provided by the installation guide. Run the executable file based on your operating system as included in the installed files. The main menu should appear as shown in the following.

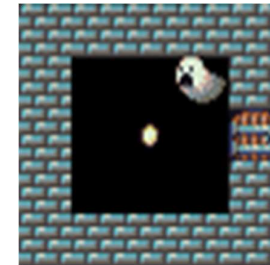


The shortcut for starting a game is pressing the Enter key on the main menu.
To quit the game, press the Escape key.

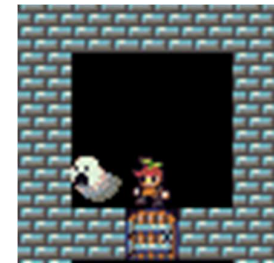
3 Playing the Game

3.1 User Interface

Once a new game is loaded, there will be a map on your screen. This is the layout of the mansion. Each time the game is loaded, or there is a level increase, the layout will change. The walls are made up of blue bricks and there are wooden doors. Each room of the mansion will have a gold coin and a ghost roaming to guard that coin.



Your character will be located randomly in the mansion.



To the top right of the screen there is a counter. This allows you to keep track of: the amount of gold you have collected; the number of steps you have taken; your score; and the current level you on. Remember – the score is based on the number of steps taken, so walk less!

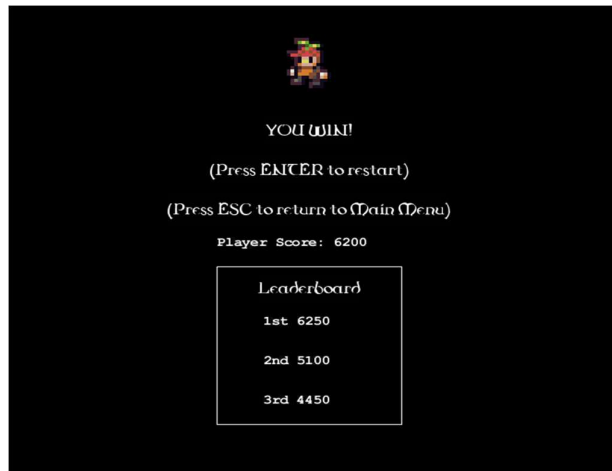
```
Gold Collected: 0
Steps Taken: 0
Player Score: 0
Current level: 1
```

3.2 Winning the game

To win the game you must complete 3 levels of the game. To complete a level, your player must move through the map to collect 3 pieces of gold in each level. After collecting the amount of gold required, a portal will open randomly on the map. You must reach the portal to win the game. Note – you might die on your way to the portal!



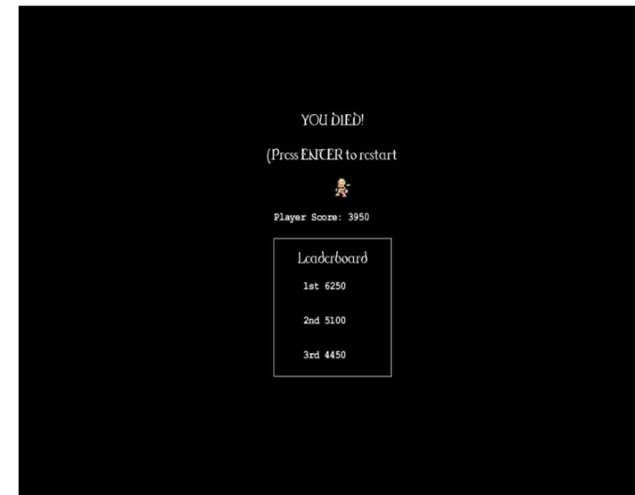
If the following screen appears, congratulations! You have won the game. You can press Enter to play again or the Escape key to go back to the main menu.



3.3 Dying in the game

You will die if you move to the same spot as the ghost in the next turn. This means that if you move to where a ghost is already there in the view, you will not die. Another special condition for when you will not die is when you spawn immediately on the same spot as the ghost at the start of a level.

If the following screen appears, oh no! You have been spooked by the ghost and turned into a mummy. No worries though, just simply press the Enter key to restart. Good luck!



3.4 Leaderboard

Attempting to place gold, silver or bronze on the leaderboard is a bonus challenge. You can choose whether to ignore, or play for, the leaderboard. To gain the high-score you will have to complete the game in as few steps as possible. Count of your steps and score can be seen in the top right hand of the screen.

4 Quitting the Game

4.1 Main menu

To quit the game, click on Quit button in the main menu.



4.2 In game

To quit during play in the game, press Escape key. Note that pressing Escape after you win the game will return you to the main menu instead.

5 Contact

We will be happy to hear your questions or comments about this game.

Email us at:

chc217@bath.ac.uk
klzc20@bath.ac.uk
ag2303@bath.ac.uk
dh731@bath.ac.uk
jl2332@bath.ac.uk
mas250@bath.ac.uk

Installation Guide for Get Rich or Die Trying

Version 1.0

15/12/17

Contributors:

Cheuk Hei Chan

Kimberley Chong

Alex Gooding

David Hurst

Jessica Lok

Megan Slattery

1.0 System Requirements	3
2.0 Installation Guide	3
2.1 Download the source code	3
2.2 Extract the files	4
2.3 Run the executable file	5
2.4 Download Eclipse or any other Java IDE (Netbeans, IntelliJ etc..)	5
2.4.1 Import the project into Eclipse or the selected IDE of your choice.	6
2.4.2 Locate the Main.Java class in main package, right click and run as 1. Java Application.	7
2.5 Play the game :)	8

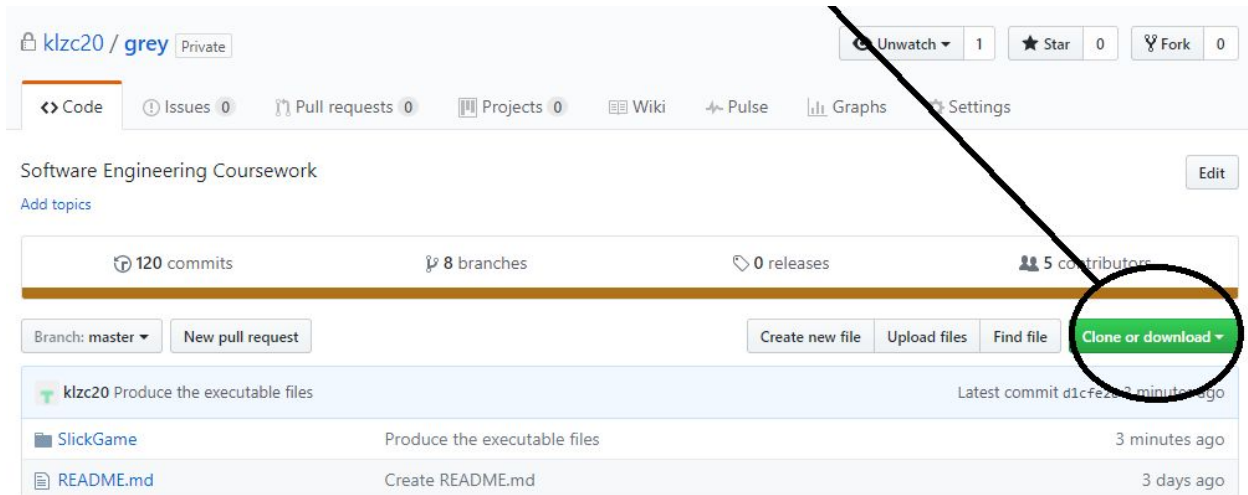
1.0 System Requirements

- Operating System: Windows, macOS, Linux
- Runtime Java: Java Runtime is required. (Download at: <https://java.com/en/download/>)
- Input/Output : Mouse and Keyboard, Sound

2.0 Installation Guide

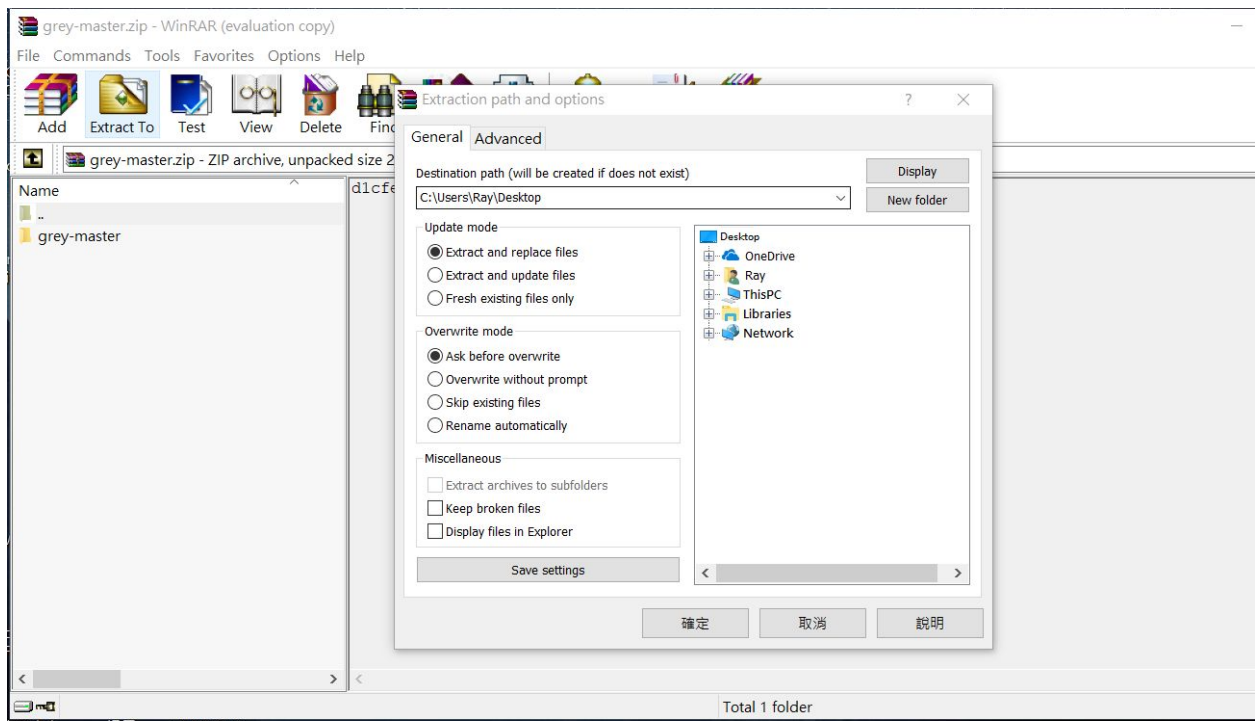
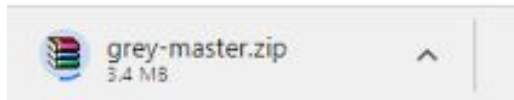
2.1 Download the source code

1. Go to the Github repository at: <https://github.bath.ac.uk/klzc20/grey>
2. Download the code from Github onto your console.
3. Make sure to download the **master** branch.



2.2 Extract the files

Unzip the file and extract to a location on your computer.



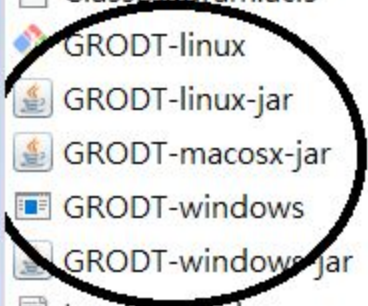
2.3 Run the executable file

Run the executable files based on your preferences and operating system located in the source file downloaded from Github.

For Windows, GRODT-windows.exe is recommended.

For Mac OS, GRODT-macosx-jar.jar is recommended.

For Linux, GRODT-linux.sh is recommended.



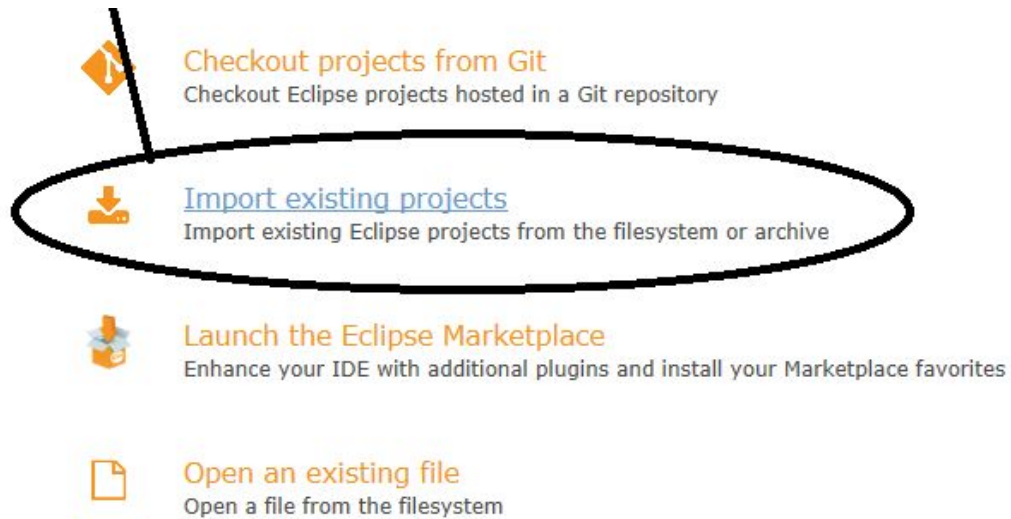
.settings	14/12/2017 14:46
lib	14/12/2017 14:46
natives	14/12/2017 14:46
res	14/12/2017 14:46
src	14/12/2017 14:46
.classpath	14/12/2017 14:46
	14/12/2017 14:46
.project	14/12/2017 14:46
ClassDiagram	14/12/2017 14:46
ClassDiagram.ucls	14/12/2017 14:46
GRODT-linux	14/12/2017 14:46
GRODT-linux-jar	14/12/2017 14:46
GRODT-macosx-jar	14/12/2017 14:46
GRODT-windows	14/12/2017 14:46
GRODT-windows-jar	14/12/2017 14:46
Leaderboard	14/12/2017 15:18
LeaderBoardTest	14/12/2017 14:46
runnablejar	14/12/2017 14:46

2.4 Download Eclipse or any other Java IDE (Netbeans, IntelliJ etc..)

Alternatively, you can open the project with a Java IDE to play the game and edit the game.

Eclipse is available at: <https://www.eclipse.org/downloads/>

2.4.1 Import the project into Eclipse or the selected IDE of your choice.



☒ Select root directory: C:\Users\david\Downloads\grey-Slick2D

☐ Select archive file:

Projects:

<input type="checkbox"/> grey (C:\Users\david\Downloads\grey-Slick2D)	<input type="button" value="Select All"/> <input type="button" value="Deselect All"/> <input type="button" value="Refresh"/>
<input type="checkbox"/> SlickGame (C:\Users\david\Downloads\grey-Slick2D\SlickGame)	

Options

☒ Search for nested projects

☐ Copy projects into workspace

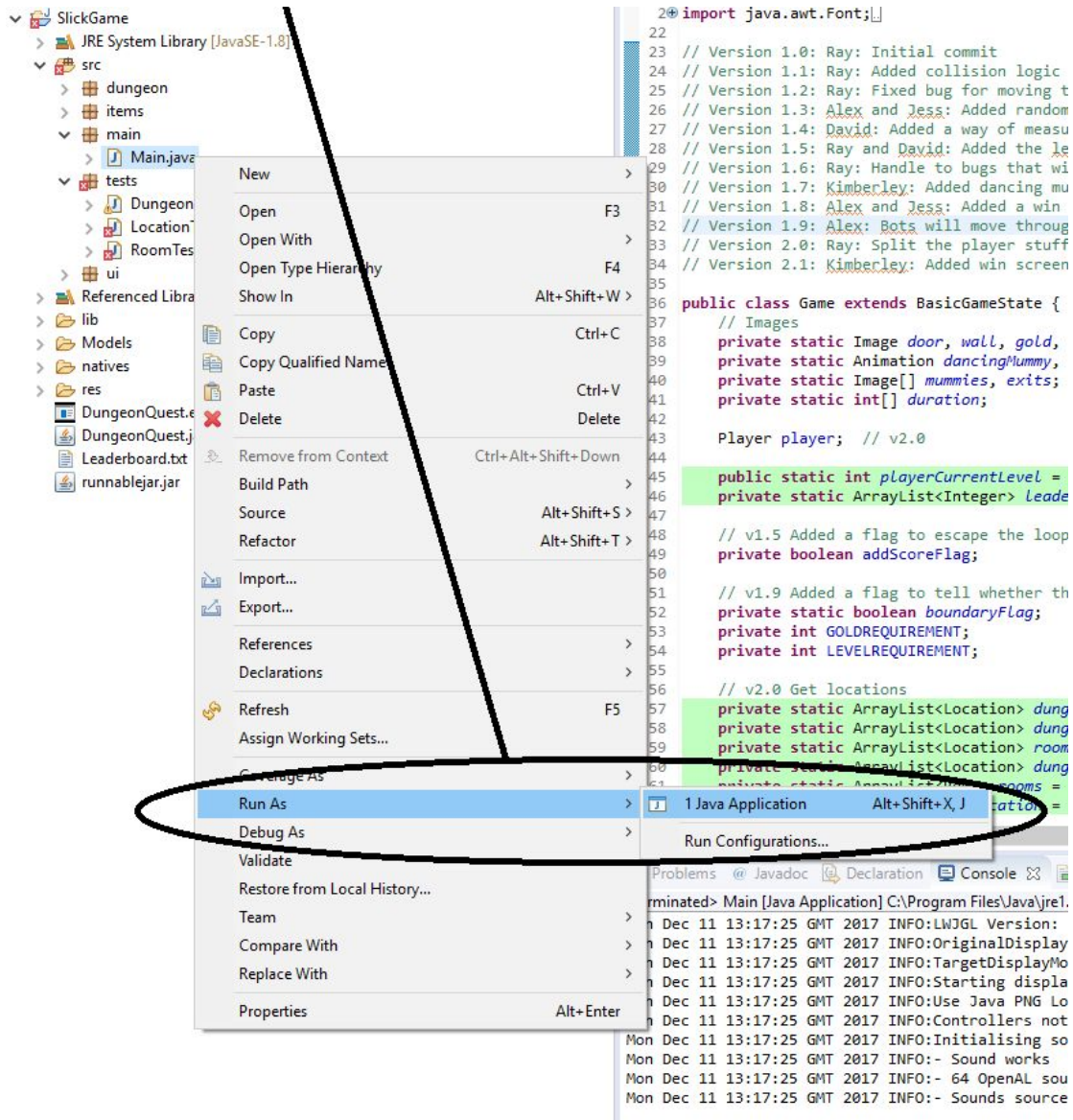
☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

2.4.2 Locate the Main.Java class in main package, right click and run as 1. Java Application.



2.5 Play the game :)



Maintenance Guide for Get Rich or Die Trying

Version 1.0

15/12/17

Contributors:

Cheuk Hei Chan

Kimberley Chong

Alex Gooding

David Hurst

Jessica Lok

Megan Slattery

1.0 Introduction	3
1.1 Purpose	3
1.2 Points of Contact	3
2.0 Project Overview	3
2.1 Description	3
2.2 Features/Rules	3
2.3 Definitions	3
3.0 Support Software	4
3.1 Eclipse IDE for Java Developers	4
3.2 Slick2D with LWJGL	4
3.3 Jarsplice	4
4.0 Skills Requirement	4
5.0 Environment Setup	4
5.1 GitHub	4
5.2 Eclipse	4
6.0 Software Maintenance Procedures	5
6.1 Eclipse	5
6.2 Jarsplice	6
7.0 System Architecture	6
7.1 MVC architecture	6
7.2 Brief introduction to Slick	7
8.0 Class Maintenance Procedures	7
8.1 Conventions / Styling	7
8.2 Class Diagram	8
8.3 Class Architecture	9
9.0 Possible Changes To The Existing Code	15
10.0 Further Development	16
10.1 Developments to The Model	16
10.2 Developments to The View	17
10.3 Developments to The Controller	18
10.4 Developments to Networking	18

1.0 Introduction

1.1 Purpose

This document provides the reader with information required to maintain or develop the game further. It provides an overview of the project covering the environment setup, system architecture, and the related classes. Further ideas are put through that were not yet developed.

1.2 Points of Contact

If assistance is required, please contact Julian Padget (j.a.padget@bath.ac.uk) at the University of Bath for further inquiries.

2.0 Project Overview

2.1 Description

A roguelike game on desktop platform based on dungeons and golds. It is a single player game with a leaderboard that saves locally to the system.

2.2 Features/Rules

- The main gameplay consists of a dungeon of an arbitrary size consisting of rooms.
- Each room has a gold coin placed at a random location in the room and a bot moving at random direction
- Both the player and the bots move onto a tile each turn.
- A bot will kill the player if both the player and bot moves to the same tile in their turn.
- A bot will not kill the player if they are already on the same tile
- A player can move between rooms through a passage represented by a door
- Each room has surrounding walls to identify it as a room
- The player will be moved to the next level after collecting the required amount of gold on that level
- A player can quit the game during play.
- The main objective of the game is to reach the maximum level with the least number of steps to score the highest points.
- The score will be compared to the leaderboard and overwrites it if a new high score is achieved.

2.3 Definitions

Player: The client represented by an avatar

Bot: A ghost that kills the player if the player moves to where the ghost is in the next move

Rogue-like: A game that take turns. Each step the player takes is counted as a turn

Dungeon: A certain level of the map with randomly generated rooms

Room: Contains bots and gold coins, allowing the player to move between them

Gold: Required to move to the next dungeon

Wall: Surrounds a room to show where each room boundaries are

3.0 Support Software

3.1 Eclipse IDE for Java Developers

Version: Oxygen.1a Release (4.7.1a). Java SRE System Library: Java SE-1.8

Purpose: Java development environment that allows change of code, testing, as well as version control via import of project from GitHub.

3.2 Slick2D with LWJGL

Slick build 237. LWJGL version 2.9.2

Purpose: Java library that allows 2D graphics and game container states for ease of game development. Contains classes related to user interface of the game.

3.3 Jarsplice

Version: 0.40

Purpose: To convert the executable jar file into an .exe file or jar files for other operating systems.

4.0 Skills Requirement

The system would require knowledge in the following languages:

- Git
- Java

5.0 Environment Setup

5.1 GitHub

1. Login to Github Enterprise using <https://github.bath.ac.uk> and sign in with your username and password
2. Clone the branch onto your computer and remember where you saved it. This will be where the files will be each time you change between branches
3. Pull the *master* branch.

5.2 Eclipse

1. Select Import > Projects from Git
2. Existing local repository > Next > Add
3. Add Git repositories > Select the GitHub repository folder > Finish
4. Select the repository > Next > Import Projects from Git
5. Import using the New Project wizard > Finish
6. New Project > Java > Java Project > Next

7. Uncheck Use default location > Location: (Choose the *SlickGame* folder in *Github* folder) > Ok > Finish

6.0 Software Maintenance Procedures

6.1 Eclipse

Issue: Wrong version Java Runtime

Error:

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.lwjgl.LWJGLUtil$3
(file:/C:/Users/Documents/GitHub/grey/SlickGame/lib/lwjgl.jar) to
method java.lang.ClassLoader.findLibrary(java.lang.String)
WARNING: Please consider reporting this to the maintainers of
org.lwjgl.LWJGLUtil$3
WARNING: Use --illegal-access=warn to enable warnings of further
illegal reflective access operations
WARNING: All illegal access operations will be denied in a future
release
```

How to fix:

1. Run > Run Configurations
2. JRE tab > Execution Environment: JavaSE-1.8
3. Apply

Issue: Wrong main class in Run configuration

Error:

Could not find or load main class main

How to fix:

1. Run > Run Configurations
2. Main class: main.Main (if package name or class name is refactored, change accordingly)

Issue: Cannot run the file on system due to difference in operating system

Error:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: no
lwjgl64 in java.library.path
    at java.lang.ClassLoader.loadLibrary(Unknown Source)
    at java.lang.Runtime.loadLibrary0(Unknown Source)
    at java.lang.System.loadLibrary(Unknown Source)
    at org.lwjgl.Sys$1.run(Sys.java:72)
    at java.security.AccessController.doPrivileged(Native Method)
    at org.lwjgl.Sys.doLoadLibrary(Sys.java:66)
    at org.lwjgl.Sys.loadLibrary(Sys.java:87)
    at org.lwjgl.Sys.<clinit>(Sys.java:117)
    at org.lwjgl.opengl.Display.<clinit>(Display.java:135)
    at
org.newdawn.slick.AppGameContainer$1.run(AppGameContainer.java:39)
```

```
        at java.security.AccessController.doPrivileged(Native Method)
        at
org.newdawn.slick.AppGameContainer.<clinit>(AppGameContainer.java:
36)
        at main.Main.main(Main.java:48)
```

How to fix:

1. Right click on SlickGame > Build Path > Configure Build Path
2. Native library location > Edit...
3. Location Path: Workspace > Slick Game/natives > (Choose appropriate operating system folder) > Ok > Apply

6.2 Jarsplice

How to create an executable:

1. Download JarSplice from: <http://niniacave.com/jarsplice>
2. Export the Java project on Eclipse as Runnable JAR file
3. Configure Runnable JAR File Specification to the SlickGame folder.
4. Run JarSplice
5. Add Jars > Add the Runnable JAR created from Eclipse
6. Add natives > Add the natives in SlickGame > natives > (Choose the operating system) > Select all the files in the folder.
7. Main Class > main.Main
8. Create fat jar, or the extra executable files as required.

Things to note:

Add natives - Make sure the natives selected are the ones meant to produce

Main class - Make sure that the main class is referenced correctly

7.0 System Architecture

7.1 MVC architecture

The system is based on a MVC architecture that separates the Model, View, and Controller.

The model consists of classes containing the logic behind creating the dungeon and bot movement, as well as the backend of updating the leaderboard.

The view consists of representing the visualizations to the player such as the main menu, when the player wins and dies. These are mostly found in the render methods of Slick2D library within the classes.

The controller is an Input interface found in the Slick2D library that helps to listen for user input. In the update methods of the classes, the class would update the view or send information to the mode.

7.2 Brief introduction to Slick

The Game Container is responsible for holding game states and the game window. A Game State would contain a `getID()` method and an integer assigned to it in the Main class. These game states would act like a page, having `init`, `render`, and `update` methods. In the `init` methods, it would retrieve the resources needed and define variables for the model. The `render` methods would display the visual elements in the game window such as text and images. To act upon a user input, the `update` methods listen for the user's input and based on these action, determine what to change for the user like showing player movement or a game state change from the main menu to the game.

8.0 Class Maintenance Procedures

8.1 Conventions / Styling

Naming

Classes: `class`, `aClass`

Methods: `method`, `aMethod`

Files: `File`, `AFile`

Folders/Packages: `lib`, `natives`, `packages`

Variables: `var`, `aVariable`, `CONSTANTVAR`

Indentations

Tab size: 4

Line length

Each line does not exceed 80 characters. Comments are exceptions.

Comments

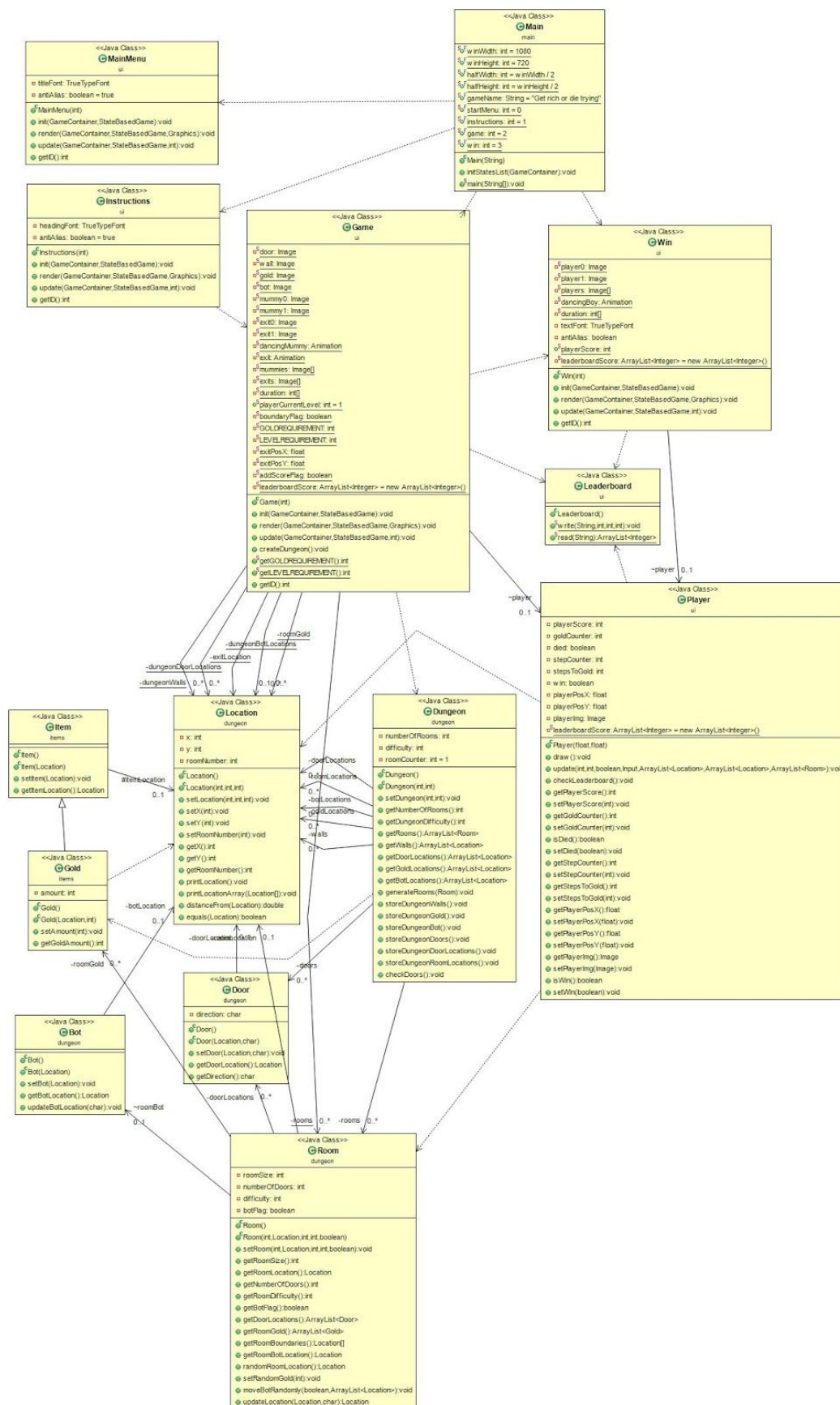
```
/*
```

```
* Block comments are used to describe the classes
```

```
*/
```

```
// Line comments are used to describe variables or show version control
```

8.2 Class Diagram



8.3 Class Architecture

```
| main  
|-- Main.java  
| dungeon  
|-- Bot.java  
|-- Door.java  
|-- Dungeon.java  
|-- Location.java  
|-- Room.java  
| items  
|-- Gold.java  
|-- Item.java  
| ui  
|-- Game.java  
|-- Instructions.java  
|-- Leaderboard.java  
|-- MainMenu.java  
|-- Player.java  
|-- Win.java
```

main package: Contains the main class for the game.

Main.java

Description: Main class for initiating the game. Set game window variables like window width, height, and name of game.

Dependencies: extends Slick2D library (StateBaseGame class)

Methods:

- `Main (String gameName)` : Default constructor that sets the game's name and add the game states: start menu and map
- `initStatesList(GameContainer gc)` : Initiation of the game states of the game
- `main(String[] args)` : Starting point of this class and the whole game. Call the Main default constructor and initial the game with the variables

Can be used to:

- Change the name of the game by changing the `gameName` variable
- Change the size of the game window by changing the `winWidth` and `winHeight` variables. Note that changing the game window size will affect most positioning of the UI elements and they would need to be adjusted accordingly
- Change to view the FPS (frame per second) of the game by changing the `setShowFPS` parameter to `true`.
- Add new game states by first assigning an integer to the class, then add the state in `Main` method as well as initializing it in the `initStatesList` method - Note that adding or changing game states will affect other game states in the classes
- Change the first game state by changing the parameter of `enterState`

dungeon package: Contains the classes responsible for the map of the game

Bot.java

Description: The class responsible for bot's movement in the game

Dependencies: (none)

Methods:

- `Bot()`: Default constructor that sets the location of the bot to a default value
- `Bot(Location botLocation)`: Places the bot at a location as specified by parameter
- `Location` takes three `int` parameters (x coordinate on the map, y coordinate on the map, room number)
- `getBotLocation()`: Returns the location of a bot
- `updateBotLocation(char c)`: Updates the location of the bot based on direction of its movement
char c is specified by 'n', 'e', 's', or 'w'

Door.java

Description: Object class responsible for setting door locations and cardinality point ('n', 'e', 's', 'w') with reference to the room.

Dependencies: Uses the location class to store (x,y) coordinates.

Methods:

- `Door()`: Default constructor that sets the doors location to (0,0) and cardinal point 'n'.
- `Door(Location doorLocation, char direction)`: Constructor that sets the doors location to doorLocation and the cardinal point to direction.
- `setDoor(Location doorLocation, char direction)`: Mutator that sets the doors location to doorLocation and the cardinal point to direction.
- `getDoorLocation()`: Accessor that returns the location of the door.
- `getDirection()`: Accessor that returns cardinal point of the door.
-
-
-

Dungeon.java

Description: The class responsible for creating the dungeon, it sets the number of rooms and the dungeon difficulty.

Dependencies: Uses the room class to store dungeon wall, door, gold and bot locations for the entire dungeon.

Methods:

- `Dungeon()`: Default constructor that sets the number of rooms to 10 and difficulty 1.
- `Dungeon(int numberOfRooms, int difficulty)`: Constructor that sets the number of rooms to numberOfRooms and difficulty of dungeon to difficulty.
- `setDungeon(int numberOfRooms, int difficulty)`: Mutator that sets the number of rooms and difficulty of dungeon.
- `getNumberOfRooms()`: Accessor that returns number of rooms in the dungeon.
- `getDungeonDifficulty()`: Accessor that returns dungeon difficulty.
- `getRooms()`: Accessor that returns an array list of rooms in the dungeon.
- `getWalls()`: Accessor that returns an arraylist of wall locations in the dungeon.

- `getDoorLocations()` : Accessor that returns an arraylist of door locations in the dungeon.
- `getGoldLocations()` : Accessor that returns an arraylist of gold locations in the dungeon.
- `getBotLocations()` : Accessor that returns an arraylist of bot locations in the dungeon.
- `generateRooms(Room initialRoom)` : Generates rooms with a random amount of doors and in a random pattern. The method starts from the initial room and places new rooms in the dungeon where each door is created.
- `storeDungeonWalls()` : Stores the boundaries of the dungeons and excludes door locations.
- `storeDungeonGold()` : Stores the locations of the gold.
- `storeDungeonBot()` : Stores the locations of the bots.
- `storeDungeonDoors()` : Stores the doors in the dungeon.
- `storeDungeonDoorLocations()` : Stores the door locations in the dungeon.
- `storeDungeonRoomLocations()` : Stores the room locations in the dungeon.
- `checkDoors()` : Checks if there are any redundant doors (leading to no rooms).

Can be used to:

- Change the algorithm of generating random rooms and associated doors by editing `generateRooms(Room initialRoom)`

`Location.java`

Description: The class responsible for setting coordinates (x,y) and room number.

Dependencies: (None)

Methods:

- `Location()` : Default constructor that sets the gold amount to 1 and the location to `itemLocatoin` from the item class.
- `Location(int x, int y, int roomNumber)` : Constructor t.
- `setLocation(int x, int y, int roomNumber)` : Mutator to set t.
- `setX(int x)` : Mutator for the x coordinate.
- `setY(int y)` : Mutator for the y coordinate.
- `setY(int roomNumber)` : Mutator for the room number.
- `getX()` : Accessor for the x coordinate.
- `getY()` : Accessor for the y coordinate.
- `getRoomNumber()` : Accessor for the room number.
- `printLoction()` : Prints a location out as a string.
- `printLocationArray(Location[] locationArray)` : Prints an array of locations out as a string.
- `distanceFrom(Location a)` : Returns the distance from a location to location a.
- `equals(location l)` : Returns whether a location is equal to location l.

`Room.java`

Description: Object class responsible for setting room parameters. Room parameters include room size, room location, number of doors, room difficulty and the bot flag.

Dependencies: `Locatoin.java`

Methods:

- `Room()` : Default constructor that sets room size to 4, location to (50,50,1), number of doors to 4 and the room difficulty to 1.
- `Room(int roomSize, Location roomLocation, int numberOfDoors, int roomDifficulty, boolean botFlag)` : Constructor that initialises (roomSize, roomLocation, numberOfDoors, roomDifficulty, botFlag) to a room, which is supplied to the function.
- `setRoom((int roomSize, Location roomLocation, int numberOfDoors, int roomDifficulty, boolean botFlag)` : Mutator that initialises (roomSize, roomLocation, numberOfDoors, roomDifficulty, botFlag) to a room, which is supplied to the function.
- `getRoomSize()` : Accessor for the room size.
- `getRoomLocation()` : Accessor for the room location.
- `getNumberOfDoors()` : Accessor for the number of doors.
- `getRoomDifficulty()` : Accessor for the room difficulty.
- `getBotFlag()` : Accessor for the bot flag.
- `getDoorLocations()` : Accessor for the rooms door locations
- `getRoomGold()` : Accessor for the rooms gold
- `getRoomBoundaries()` : Accessor for the room boundaries.
- `getRoomBotLocation()` : Accessor for the rooms bot location.
- `randomRoomLocation()` : Method to randomly generate a location within a certain room.
- `setRandomGold(int goldBound)` : Randomly generates gold in a room.
- `moveBotRandomly(boolean boundaryFlag, ArrayList<Location> altBoundaries)` : Moves the room bot one tile within the room or the given alternative boundaries.
- `updateLocation(Location l, char c)` : Updates a given location based on the direction of movement.

Can be used to:

- Change the algorithm of where a random location of a room will be in `randomRoomLocation()`
- Change the algorithm of where a random location of a gold will be in `setRandomGold(int goldBound)`
- Change the algorithm of random bot movement in `moveBotRandomly(boolean boundaryFlag, ArrayList<Location> altBoundaries)`

items package: Contains the classes of items in the game.

`Gold.java`

Description: Object class responsible for setting gold location and gold amount.

Dependencies: extends `Item.java`

Methods:

- `Gold()` : Default constructor that sets the gold amount to 1 and the location to `itemLocatoin` from the `item` class.

- `Gold(Location goldLocation, int goldAmount)` : Constructor that sets the gold location to `goldLocation` and the gold amount to `goldAmount`.
- `setGoldAmount(int goldAmount)` : Mutator to set the gold amount to `goldAmount`.
- `getGoldAmount()` : Accessor for the amount of gold.

`Item.java`

Description: Object class responsible for setting item locations.

Dependencies: `Location.java`

Methods:

- `Item()` : Default constructor that sets the item location to (0,0,1).
- `Item(Location itemLocation)` : Constructor that sets the item location to `itemLocation`.
- `setItem(Location itemLocation)` : Mutator to set the item location to `itemLocation`.
- `getItemLocation()` : Returns the item location.

ui package: Contains the game state classes and elements of the game.

`Game.java`

Description: Visualization of the dungeon and player. Player movement and collision detection

Dependencies: Extends Slick2D library (`BasicGameState` class)

Methods:

- `Game(int game)` : Default Constructor
- `init(GameContainer gc, StateBasedGame sbg)` : Initiate the map. Load images. Create Dungeon and player with coordinate stores in variables.
- `render(GameContainer gc, StateBasedGame sbg, Graphics graphics)` : Drawing the dungeon, gold, bots and player. Updates the gold amount, steps taken, score, level and checks if player still alive.
- `update(GameContainer gc, StateBasedGame sbg, int delta)` : Updating the game with user input and collision detection
- `getID()` : return the game state id

Can be used to:

- Adjust camera offsets
- Change gold and level requirements
- Change the file paths of images and leaderboard
- Change the location of the exit portal of the dungeon after getting final piece of gold
- Add or edit visual elements of the dungeon displayed in the game (including the ghosts, walls, rooms, and doors)
- Change the key of quitting the game
- Change the appearance of the screen after the player dies

`Instructions.java`

Description: Visualization of the instructions for players who do not know how to play the game

Dependencies: Extends Slick2D library (`BasicGameState` class)

Methods:

- `Instructions(int state):` Default Constructor
- `init(GameContainer gc, StateBasedGame sbg):` Initiate the instruction menu. Telling the user how to player the game
- `render(GameContainer gc, StateBasedGame sbg, Graphics g):` Displays the texts, screenshot, and back to main menu button
- `update(GameContainer gc, StateBasedGame sbg, int delta):` Update the game if user click on any buttons

Can be used to:

- Change the instructions display in `render` method
- Adjust the position of the UI elements

`Leaderboard.java`

Description: Read and write the 3 highest scores from and to the `leaderboard.txt` file

Dependencies:

Methods:

- `write(String fileName, int highestScore, int secondScore, int thirdScore):` Write the top 3 highest scores into the file
- `ArrayList<Integer> read(String fileName):` Read the scores from the file and return an array list

Can be used to:

- Change the way how the leaderboard is read and wrote
- Change the leaderboard to a server connected database
- Change the number of scores displayed on the leaderboard
- Change to a database management system instead of reading off text document

`Player.java`

Description: Contains stats and methods relating to the player in the game

Dependencies:

Methods:

- `Player(float x, float y):` Default Constructor with the positions
- `draw():` Drawing the player
- `update(int initialOffsetX, int initialOffsetY, boolean boundaryFlag, Input input, ArrayList<Location> walls, ArrayList<Location> gold, ArrayList<Room> rooms):` Tracking the user's input. Check collision. Update player's variables
- `checkLeaderboard():` Check if the user's scores is in the top three and update the leaderboard.
- Getter and setters of the variables

Can be used to:

- Change the image file path of the player sprite
- Change the movement controls of the player
- Add in networking behaviour
- Add player name to the `Leaderboard.txt` output file

- Change the behaviour of how many scores are added to the leaderboard and add names of players

MainMenu.java

Description: Start menu of the game

Dependencies: Extends Slick2D library (BasicGameState class)

Methods:

- MainMenu(int state): Default constructor
- init(GameContainer gc, StateBasedGame sbg): Initiate the start menu
- render(GameContainer gc, StateBasedGame sbg, Graphics g): Rendering the buttons(Start Game & Quit) in the start menu
- update(GameContainer gc, StateBasedGame sbg, int delta): Update the game if user click on any buttons
- getID() : return the game state id

Can be used to:

- Add more buttons for different states
- Change the background music.

Win.java

Description: Visualization of the game state after the player has won the game

Dependencies: Extends Slick2D library (BasicGameState class)

Methods:

- Win(int state): Default constructor
- init(GameContainer gc, StateBasedGame sbg): Initiate the win menu
- render(GameContainer gc, StateBasedGame sbg, Graphics g): Display the message for winning the game and leaderboard
- update(GameContainer gc, StateBasedGame sbg, int delta): Listen for user input
- getID() : return the game state id

Can be used to:

- Change the appearance of the screen after the player wins

9.0 Possible Changes To The Existing Code

Game.java

The Game class contains the gold and level requirements. Once they are changed, the difficulty for the player may be increased or decreased as well. The `GOLDREQUIREMENT` changes the number of gold required to proceed to the next level. This variable is constant in each level. The `LEVELREQUIREMENT` changes the number of level the player has to go through to win the game changes. More properties can also be added such as player's life, inventory, other items such as weapons and other elements to make the game more fun. The conditions of winning the game and dying in the game can also be changed in this file. What happens after the death of a player is edited in the file as well.

Room.java

The Room class has the most flexibility to change the game within it. Firstly, a room does not have to contain a bot. If `botFlag` is changed to false, then the room will not generate a bot.

The size of a room can change by the way of the `roomSize` parameter. Currently the size of our rooms in the dungeon is 4, which generates rooms with 5x5 wall dimensions and 3x3 room tile dimensions that the player can stand on.

The `difficulty` parameter can also be altered. At the moment it is set to 1 but can be changed to another integer. Currently this will only have an impact on the `goldAmount` variable - with a higher difficulty, fewer pieces of gold will spawn in the room. The formula that choose the amount of gold within the room can be changed if desired. This will make the overall game easier or harder regardless of what the difficulty of the room is.

The `moveBotRandomly(boolean boundaryFlag, ArrayList<Location> altBoundaries)` method has the ability to let the bot move completely free of any boundaries when `boundaryFlag` is equal to false and the `altBoundaries` are empty. When `boundaryFlag` is true, which it is for the majority of the game, the bot will always be contained within the walls of its room. The `altBoundaries` parameters allows for any set of Locations to be the boundaries for a bot.

`Dungeon.java`

Another big class that could be altered in its current state is `Dungeon`. This class mainly contains information about the dungeon as a whole.

Similarly to the Room class, the difficulty of the dungeon can be changed. This parameter currently has no real use but could be implemented in the future.

Within the `generateRooms(Room initialRoom)` method, there is flexibility to change the distribution of the number of doors to be generated within the next room. This along with changing the number of doors the initial room has, can drastically alter the likelihood of bigger or smaller dungeons being spawned.

10.0 Further Development

10.1 Developments to The Model

The game created serves its purpose well but is simple in design. The game could take on a different shape with development to the logic behind the game. The perhaps most obvious addition to the model of the game is to introduce new items. This could be done by extending `Item.java` in a similar way to `Gold.java`. Possibilities can include but are not limited to: adding weapons for the player to use, adding health for the player to pick up and adding stationary obstacles for the player to move around. These additions may require restructuring of the game in other areas. For example, to include all these new items it may be sensible to use larger rooms to house them. All of these items would need to be stored in lists in the `Dungeon` class and accessed in the `Game` class to implement

collision logic for them. Adding to this, the health items could increase the player's health which would in turn need to be stored in the Player class.

The dungeon difficulty parameter could be utilised to control a number of aspects of the game. It simply could be used to control the ratio of room difficulties - a higher dungeon difficulty would result in a higher proportion of difficult rooms. It could be used to also increase the size of the dungeon relative to the difficulty. Since the Dungeon constructor contains a number of rooms parameter, it would be easy to implement this.

For more abstract ideas of what the game could be, changes could be made to the dungeon generating algorithm within the Dungeon class. At the moment room sizes for newly generated rooms are fixed to a constant value. This could potentially be changed so that the map is made out of a combination of rooms of various sizes. However, a lot of thought will have to go into the logic behind this process as it is not immediately obvious how N rooms of various dimensions would fit together without overlap.

Changes could be made to the bot so that it has more functionality. For example, instead of the bot being designed to simply end the game when it comes into contact with the player, it could instead steal the players gold and only once the players gold count is 0 can the play die. This would be implemented by changing the bot logic with the Game class. This doesn't have to necessarily replace the old design of bot - by extending a new class called NPC, we could have multiple different types of bot with different functionality. A bot that kills you and a bot that steals your gold are two such examples.

Finally, the leaderboard does not have to be hosted locally. Instead, player's scores can be saved on an online database. This would require a substantial amount of new code to be written for constructing the database and hosting it online. Knowledge of MySQL and PHP would be useful for this task.

10.2 Developments to The View

The graphics are clear and coherent - they do their job well and blend together consistently to meet the style of the game. However, for the most part they are fairly static with few changes to them once the game has started, apart from the images moving to new coordinates. Lots of changes could be made to the visualisation of the game to make it more interesting to play.

One of the things that stands out about the game currently is the lack of variety in the sound design. Sound effects for various elements of the game were created but were not implemented in the end. Currently the theme tune for the game loops indefinitely as soon as the main menu is opened. It may be preferable to still have this happen upon opening the game but the theme tune could stop once the main menu is left. Background ambience may be more suitable to loop whilst the game itself is being played with short sound effects for things like collecting gold and contact with bots. These will most likely have to be implemented within the logic for the corresponding collision in the Game class. A sound effect could also be added for when the player wins the game within the Win class.

It would also be useful if the game had some kind of master volume control since the theme music is fairly loud at the moment. This could be implemented by adding a setting menu that can change the volume based on user input. It can then call for a method known as `setVolume` from the `Music` class in the `Slick` library.

For some of the possible developments to the model of the game, extra menu options would need to be added to the view. If various difficulty levels of the game were to be added, there would need to be a submenu generated from the `MainMenu` class to allow users to select from a list of difficulties. Perhaps there could be similar options for the size of the rooms generated or number of bots. However, most likely these factors could all be dictated by the `dungeon difficulty` parameter.

An initial feature was to have a minimap in the game for the player to be able to locate bots and golds from within the map but this happened to be not necessary as majority of the dungeon could be seen from within the game window. A possible change to this is to scale the sprites to a bigger size and adjust the sprites to be apart from each other so that they do not overlap. The images also needs to be filtered such that the quality would not be reduced after being scaled. The images can be scaled such that from the game window, only one room is visible. Then, a mini map can be implemented to show the position of the player and all the rooms. The room of which the player is in can also be highlighted in the mini map such that the player can find the way between rooms.

To enhance player experience, animated transitions between the dungeons can be used such that the player would exit from a door and come in from a particular door in a new level. This can be done by assigning condition variables for when a player has finished the level and stop the user input. Then, the player can be directed to the exit point via calculated linear vector distances to get to the exit point on its own. When moving, a sprite sheet can be used for the bots such that their movements are like a gif or to show direction of movement.

10.3 Developments to The Controller

Changes to the way the player controls the game are fairly dependent on further developments being made to at least the model of the game. In its current state the game doesn't have much need to change or add to the input controls. However, if for example attack logic for the player was added, it may be sensible to map an attack button on the keyboard for the user. This would be the same with any action that the player can decide to do within the game apart from simply moving. These changes would need to be made within the `Player` class, as that is where user input is recorded.

10.4 Developments to Networking

To implement a multiplayer game, the `JagNet` (Java Game Networking) library can be used. This would require advanced knowledge in TCP protocols and firewalls. The game would need to connect to a server and a database with networking behaviours for multiplayer functionality. With a database and server, the game stats of the players such as high scores and saved game states can also be stored

remotely. That way, the player can come back to the saved level on any other devices. The possible challenges in implementing a Player versus Player mode would be having a split screen as well as working out a condition when a player has already moved to the next level but another player is still in the previous level.

If networking behaviour is implemented, it would be ideal for player names to be visually presented so that the players can identify each other on the map. Furthermore, it will provide a challenge for players trying to beat the leaderboard and putting their name there.