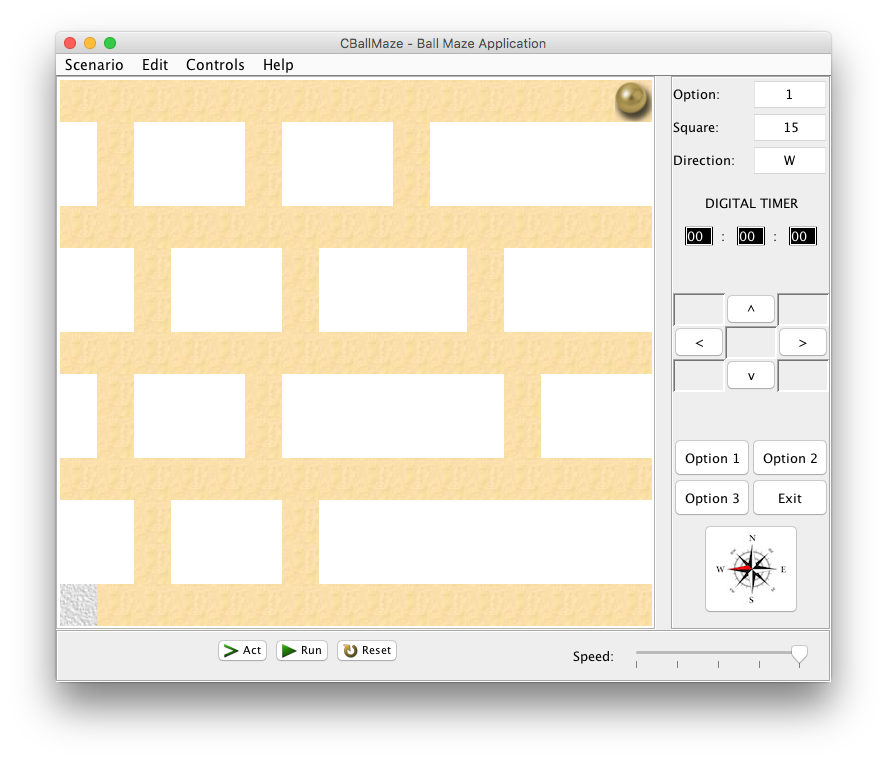
**CSY1020: Problem Solving & Programming**

**Assignment 2: Programming (Java) (50%)**

Ana Rosa Neto Gaspar

18400473

BSc Computing



University of Northampton

Submission: **Sunday, 22nd April 2018, by 23h59.**

**TABLE OF CONTENTS**

University of Northampton 1

1. INTRODUCTION & PROBLEM SPECIFICATION 3

1.1. Aim 3

1.2. Objectives 3

2. ANALYSIS 5

2.1. System Requirements: Essential (Graphical User Interface) 5

2.2. System Requirements: Additional (Functionality & Complexity) 5

3. DESIGN 7

3.1. GUI Design 7

3.2. Design (Pseudo-code/Flowcharts) 7

4. IMPLEMENTATION 8

5. TESTING 9

System Requirements: Essential (Graphical User Interface): 9

System Requirements: Additional (Functionality & Complexity): 9

6. CONCLUSION & RECOMMENDATIONS 10

6.1. Introduction 10

6.2. System Requirements: Essential (Graphical User Interface) 11

6.3. System Requirements: Additional (Functionality & Complexity) 11

REFERENCES 13

BIBLIOGRAPHY 13

**TABLE OF FIGURE**

Figure 1.1: Emulation - CBabyBallBounce.java Application – Opening State MacOSX. 3

Figure 3.1: Emulation - CCarCrash.java Application – Opening State MacOSX. 7

Figure 6.1: Emulation - CBabyBallBounce.java Application – Opening State MacOSX. 10

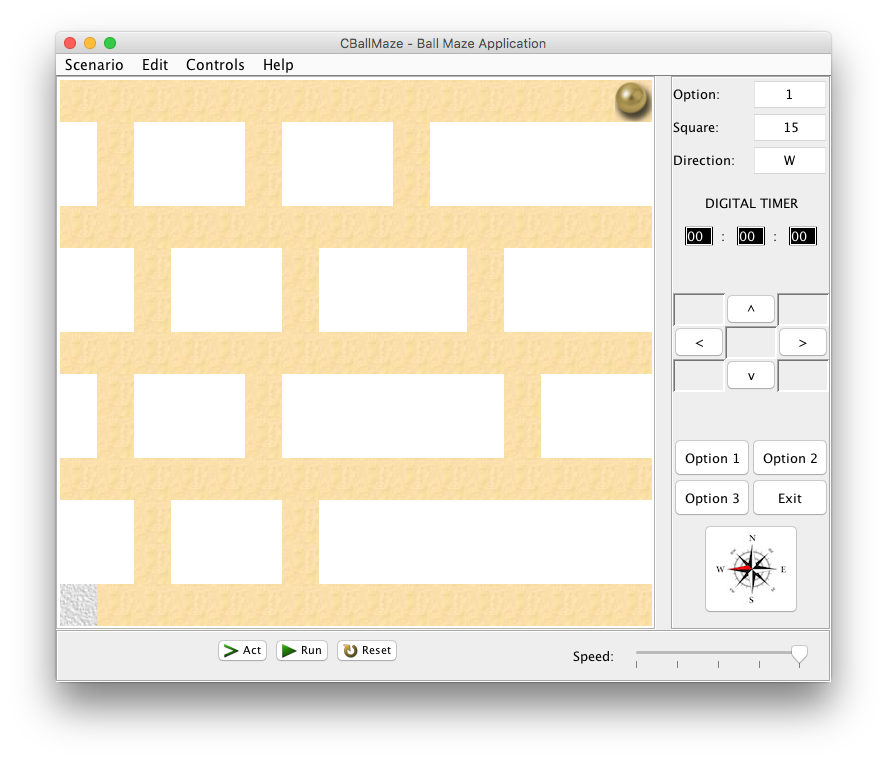
**LIST OF TABLES**

# INTRODUCTION & PROBLEM SPECIFICATION

## 1.1. Aim

Using the Green Scenario included, produce a technical report and accompanying Java GUI application, simulating control of the ball moving (navigated) around the maze using left, right, up and down buttons/keys.

The problem is designed to be open within the rules below, to enable you to develop **your solution(s) to the problem**.

**

*Figure 1.1: Emulation - CBabyBallBounce.java Application – Opening State MacOSX.*

## 1.2. Objectives

**Rules (Basic)** Create a simulation of the ball moving around the pitch, where:

* The ball must only travel when on the ‘sand’-coloured blocks otherwise it should not move.
* The ball must move one whole ‘sand’-coloured blocks at a time every time a movement key is pressed - via a direction button (<, > v ^)) - (when movement is possible).
* Must use the scenario provided.
* Must stop when it the ball reaches the grey block at the end of the maze.
* The basic solution must be completed using the ‘act’ button (accessing the **moveBall()** method within the **CBallMaze.class**).

**Rules (Intermediate and advanced)** Create a simulation of a ball moving around the maze, where:

Rules (Intermediate)

* Whilst maintaining the features of the basic solution add the following
  + When there is a block below the block the ball is automatically go down. In other words if the ball can drop it ‘falls’ down until a white space is below it.
  + Add a sound effect when the ball drops.
  + The ball must not fall into the white spaces.

Rules (Advanced)

* + For higher grades on the solution part of the assignment see the marking scheme/rubric. You must NOT change the layout and all changes should still meet the criteria of **Rules (Basic**).

# ANALYSIS

## 2.1. System Requirements: Essential (Graphical User Interface)

* 13 x 16 grid of **JButton**’s or Icon’s.
* 4 **JButton**’s for the game options ‘Option 1, Option 2, Option 3’ and ‘*Exit*’.
* 3 **JButton**’s for ‘Act’, ‘Run’ and ‘*Reset*’.
* 9 **JButton**’s for ‘*Forward >*’, ‘Backwards <’, ‘Up ^’, ‘Down v’ should move the ball in the appropriate direction by one square for each press (plus 5 blank).
* The compass icon (**JButton)** should illustrate the current direction for the ball.
* **3 JLabel**’s for ’Option’, ‘*Square*’ and ‘*Direction*’.
* **3 JTextField**’s for the current ‘Option’, Location/*’Square’* and *‘Direction’* of the ball. Use the square identification method e.g. 0 to 207 and N, E etc.
* 3 **JLabel’s** for the ‘DIGITAL TIMER and the two :’, with 3 **JTextField’**s for the hours, minutes and seconds.
* Create a **JFrame** application, which opens to the set size (775 \* 650).
* **JFrame** title set as "*CBallMaze – Ball Maze Application*".

## 2.2. System Requirements: Additional (Functionality & Complexity)

* Application icon for the **JFrame** used (Windows only).
* Application dock icon.
* The ‘Run’ **JButton** should show the ball moving between the continuously from the initial position (Option 1 – default opening state – ball top right-hand corner) to the end position at the grey square/tile (bottom left-hand corner.
* The ‘Reset’ **JButton** should clear/reset the application to its starting/default opening state.
* The ‘Act’ **JButton** should step through the above ‘Run’ sequence one move at a time.
* Discuss and implement the different options for the 3 configurations.
* The ‘Option 1, Option 2, Option 3’ **JButton**’s should display different tile/object configurations/locations.
* A **JMenuBar** could be included with **JMenu**’s for the *Scenario, Edit, Controls* and *Help*, which include **JMenuItem**’s of *Exit (Scenario)*, *Help Topic* and *About (Help)*.
* Additional **JButton**’s may be used to improve the applications usability e.g. ball movement – in random/predefined direction, jump objects/obstacles in Option 2 or 3 etc.
* The ball drops down the maze.
* A sound effect is heard when the ball drops down to the next level.
* Create a **JFrame** application, which is not resizable.
* Create a **JFrame** application, which centres itself on the monitor.
* Discuss the possibilities for incorporating intelligence/checks for whether moves are valid.
* Digital Timer should start and stop when ‘Run’ is pressed and stopped when a ball gets to the end.
* Implement intelligence/checks for whether moves are valid.
* A **moveBall()** method should be used to solve the problem. The **moveBall()** method should include **move(MOVE\_LEFT), move(MOVE\_RIGHT), move(MOVE\_UP), move(MOVE\_DOWN)** methods (see below).

public void moveBall()

{

move(MOVE\_LEFT);

……………….

move(MOVE\_RIGHT);

}

The applications **must** be demonstrated (see below). The source code file containing the **main()** method and the compiled byte code **class** files should be named as follows:

**CBallMaze.java** & **CBallMaze.class**

# DESIGN

## 3.1. GUI Design

*Figure 2: Graphical User Interface Design*

## Design (Pseudo-code/Flowcharts)

Frame creation

* Size 775x650 (resize disable)
* Title “CBallMaze – Ball Maze Application”
* Set frame Icon
* Display location (center of screen)

Frame components creation

* Main JPanel
  + Set GridLayout
  + Set size
  + Add to frame
* Right JPanel
  + Set size
  + Add to frame
  + Components
    - Outputs JPanel
      * Set GridLayout
      * Set size
      * Add option, square and direction labels and text fields
    - Timer JPanel
      * Set size
      * Add timer labels; hours, minutes and seconds text fields
    - Arrows keys JPanel
      * Set GridLayout
      * Set size
      * Add 9 JButtons (4 with direction arrows, 5 disable)
    - Options JPanel
      * Set GridLayout
      * Set size
      * Add 4 JButtons
    - Direction JPanel
      * Set size
      * Add direction ImageIcon
* Bottom JPanel
  + Set size
  + Add to frame
  + Components
    - Act JButton
    - Run JButton
    - Reset JButon
    - Speed JLabel
    - JSlider

Run

Display GUI in the centre of screen

IF arrow keys panel buttons pressed

* button ‘^’
  + Check current position
  + Check if next position (square -16) is a sand block
  + If true move up one time; update square and direction text field; update direction and main panel;
  + Else don’t move.
* button ‘<’
  + Check current position
  + Check if next position (square -1) is a sand block
  + If true move left one time; update square and direction text field; update direction and main panel;
  + Else don’t move.
* button ‘>’
  + Check current position
  + Check if next position (square +1) is a sand block
  + If true move right one time; update square and direction text field; update direction and main panel;
  + Else don’t move.
* button ‘v’
  + Check current position
  + Check if next position (square +16) is a sand block
  + If true move down one time; update square and direction text field; update direction and main panel;
  + Else don’t move.

IF options panel buttons pressed

* ‘Option 1’ button
  + Update main panel to default model; Set outputs panel to default;
* ‘Option 2’ button
  + Switch main panel model; Update outputs panel.
* ‘Option 3’ button
  + Switch main panel model; Update outputs panel.
* ‘Exit’ button
  + Close the frame

IF ‘Act’ button pressed

* Check current position
* Check if next position (square +16) is a sand block
* If true move down till next position isn’t a sand block; update square and direction text field; update direction and main panel;
* Else Check if next position (square -1) is a sand block
* If true move left one time; update square and direction text field; update direction and main panel;

IF ‘Run’ button pressed

* Do one click on ‘Reset’ button
* Get slider current value
* Set and Start timer
* While timer is running
  + Checks slider value
  + Do clicks on ‘Act’ button´
* If ball position == 192
  + Timer stop;
  + Disable arrow keys panel and ‘Act’ buttons

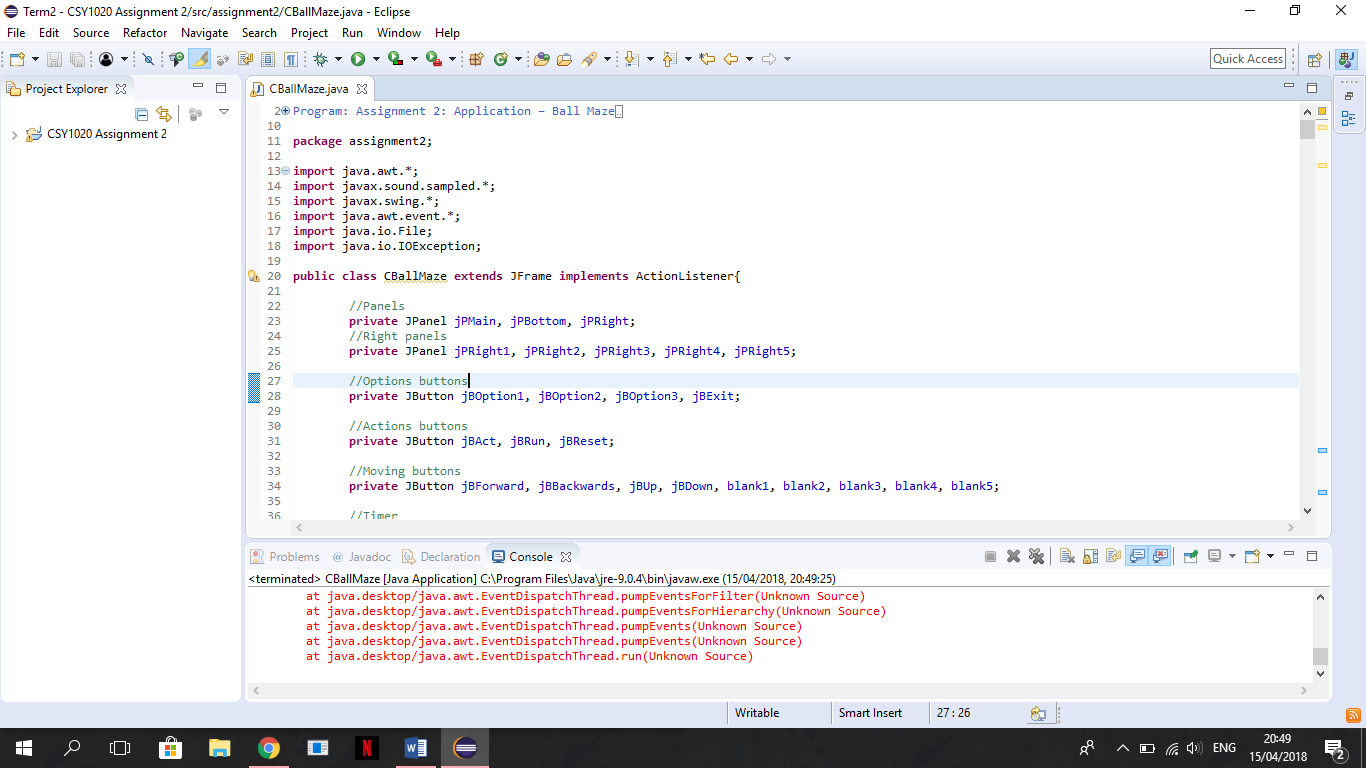
IF ‘Reset’ button pressed

* Enable arrow keys panel and ‘Act’ buttons
* If timer is running
  + Set output panel text fields and direction image to default
  + Stop timer
  + Set timer panel text fields to 0
  + Set main panel to default (ball position = 15)

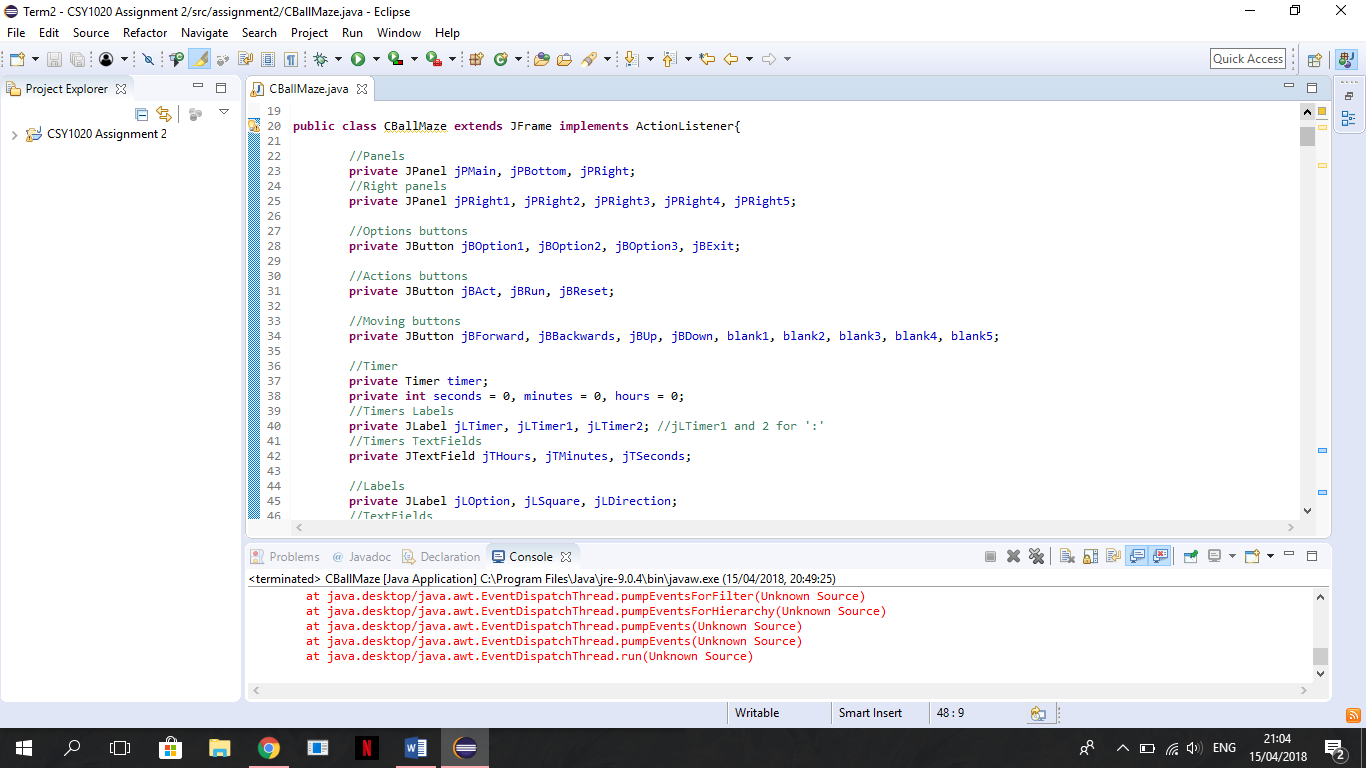
IF speed controller changed

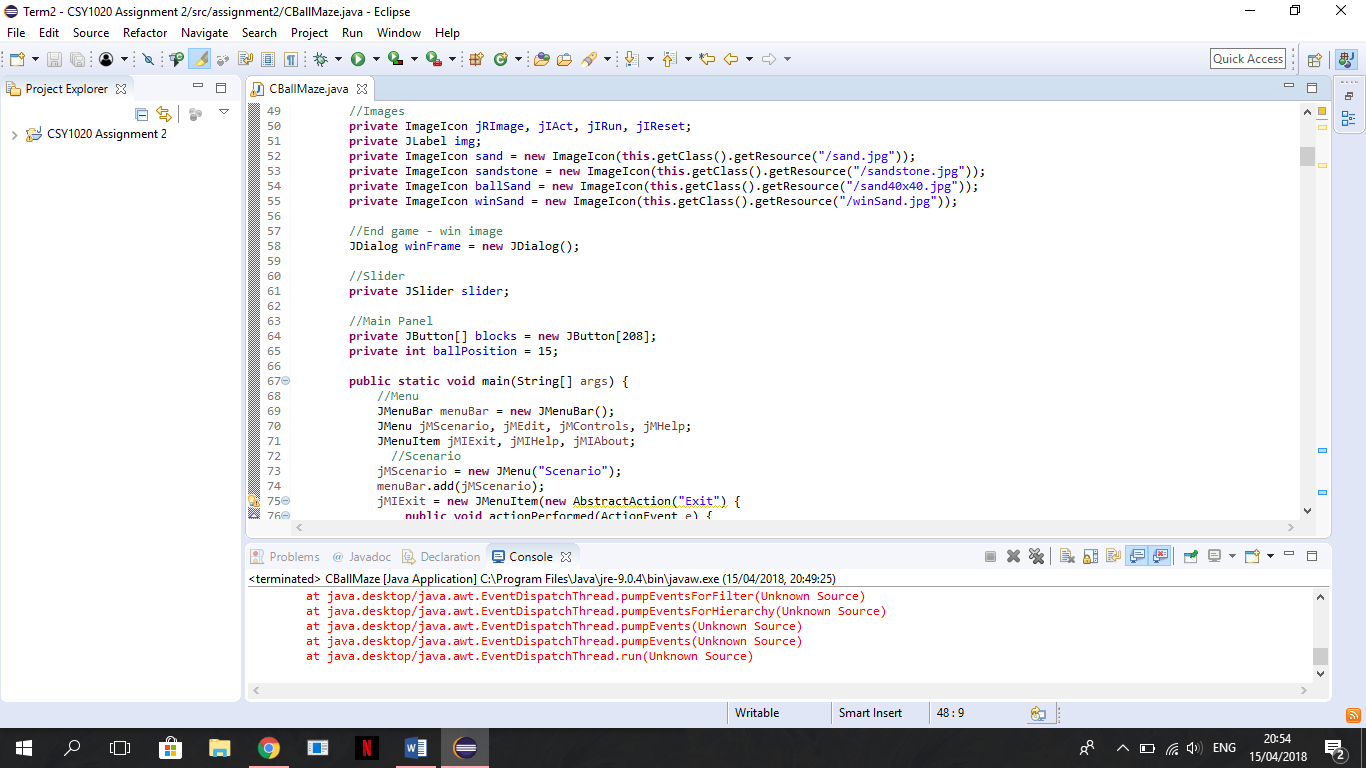
* Change ball movement speed

# IMPLEMENTATION

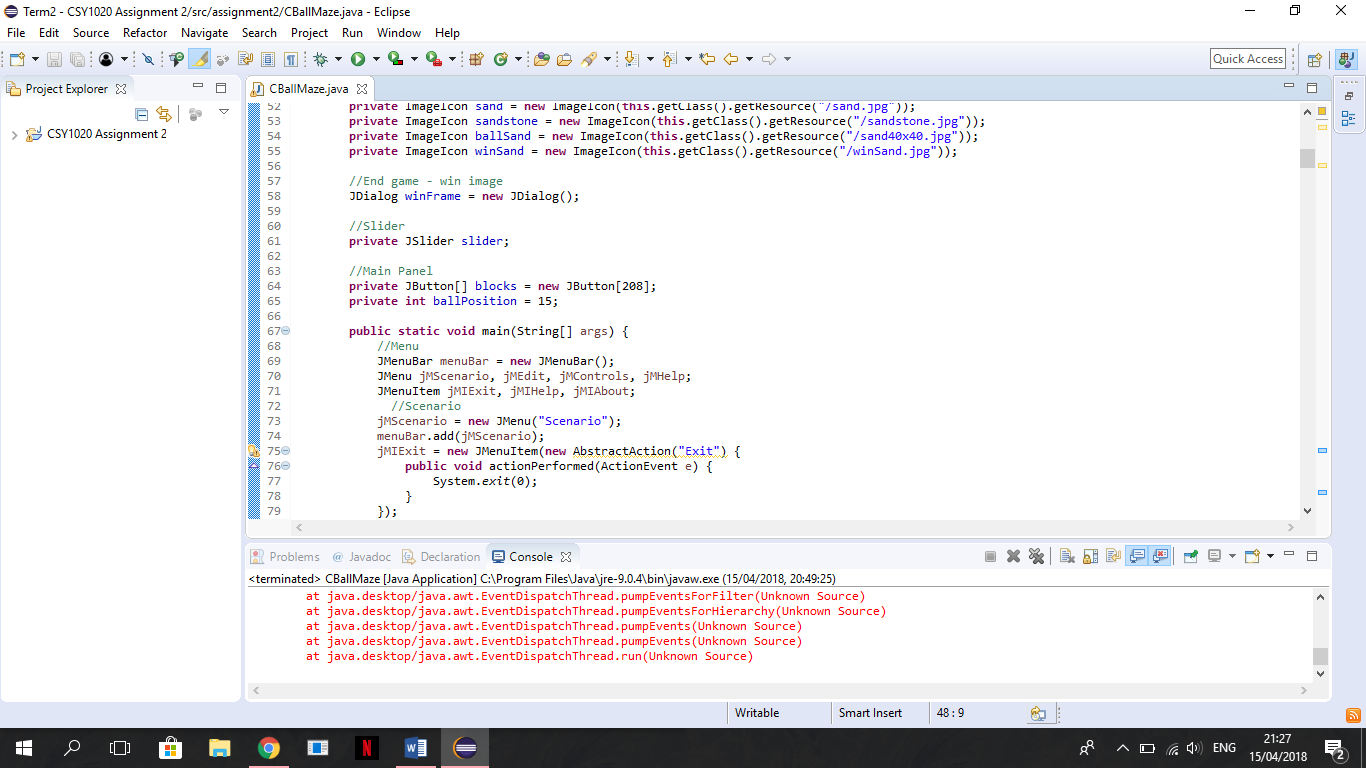
The code below allowed the writer to access to the named components. If these were not imported at the start of the application then it would not be possible to use the Java features.

*Figure 3: Imports*

The variables used for the application features (panels, buttons, text fields, slider, images) should be named and initialised after the class name has been defined. These variables are the ones we use more than one and in different parts of the code.

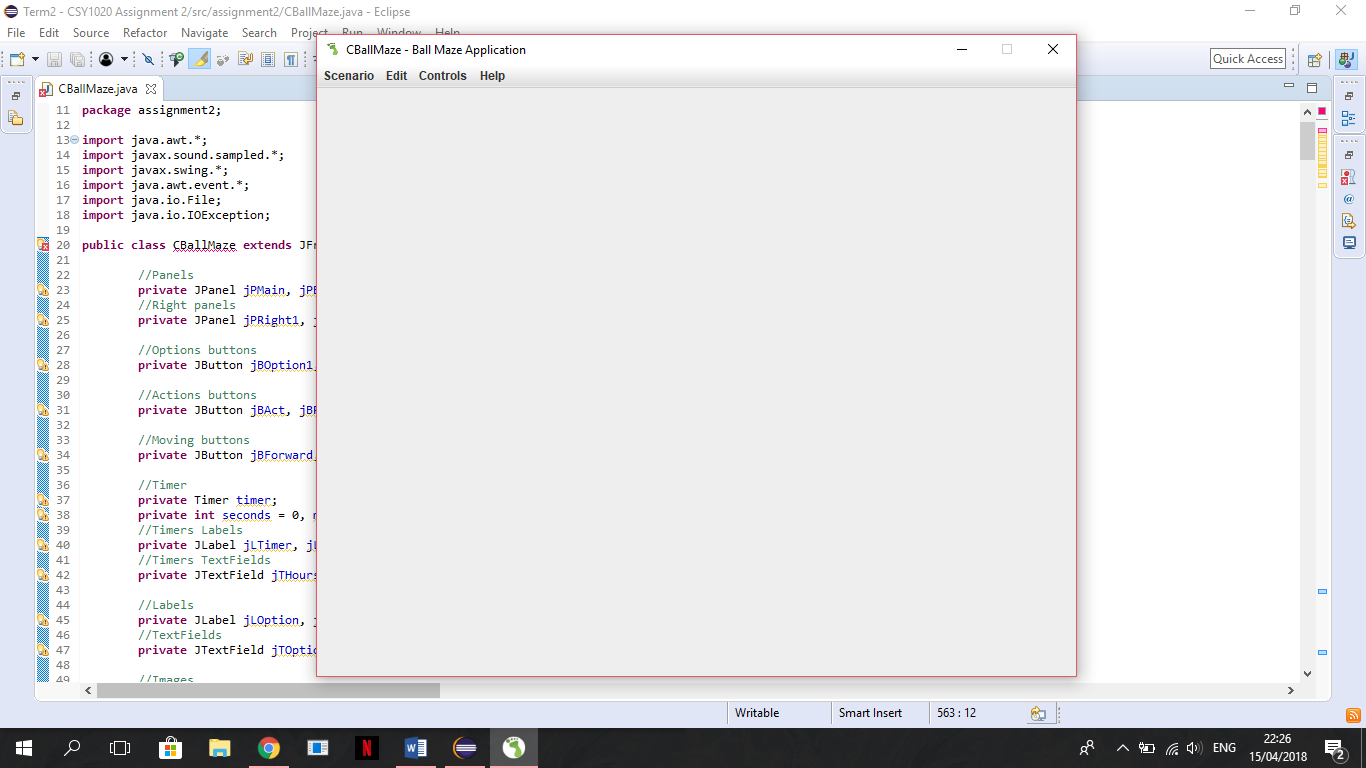


*Figure 4: Variables*

The game most important variable is defined as an array of buttons. This array is important to define where the game images are going to be.

*Figure 5: Buttons array*

The code and image below shows the creation and result of the main method. This method is the entry point for the application and will subsequently invoke all the other methods required by the program. In this method the frame is created.



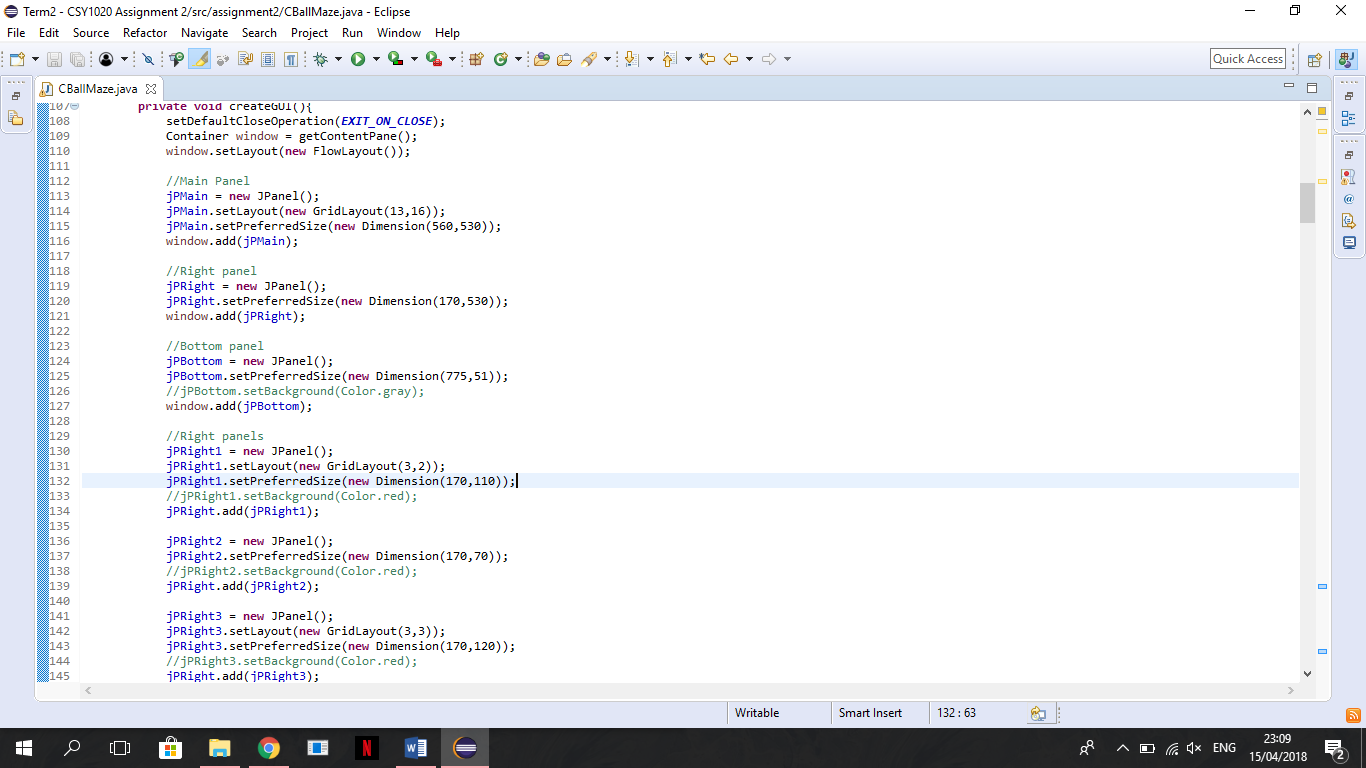
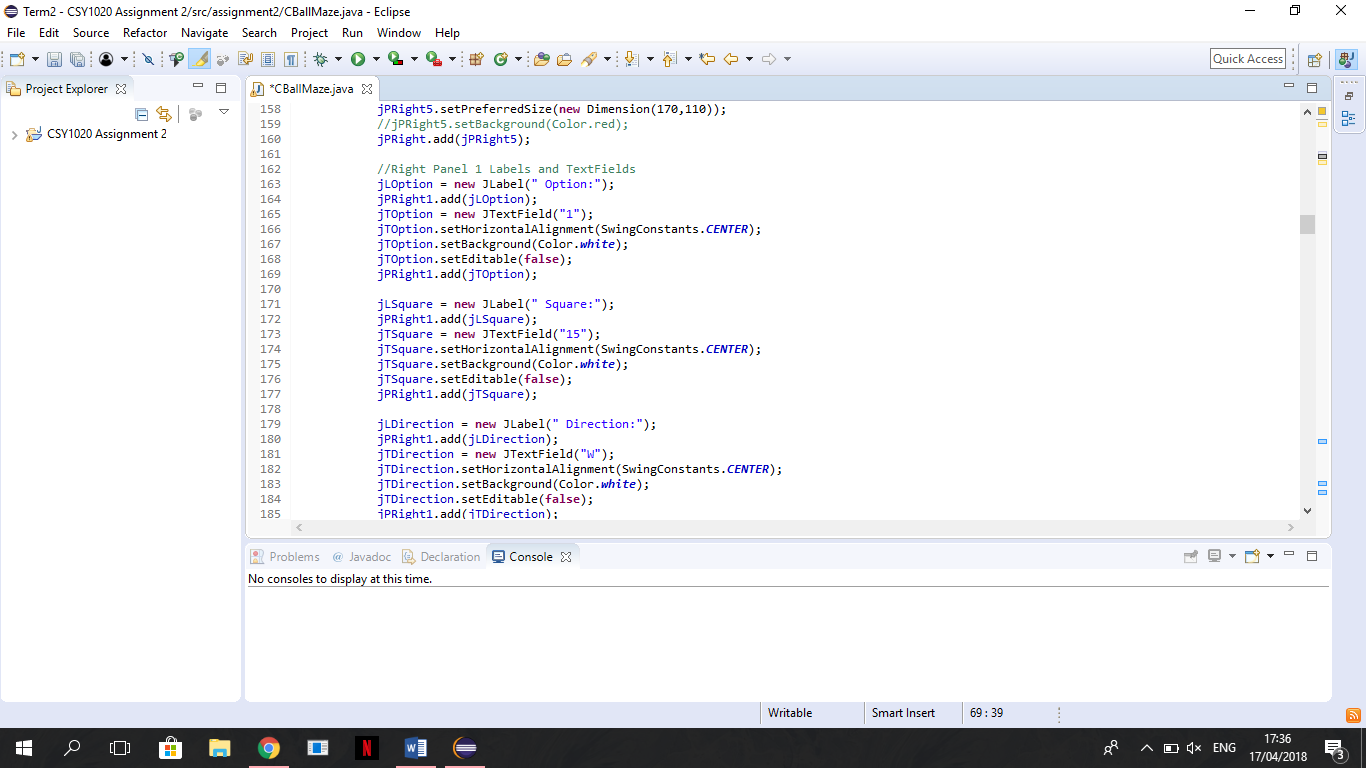
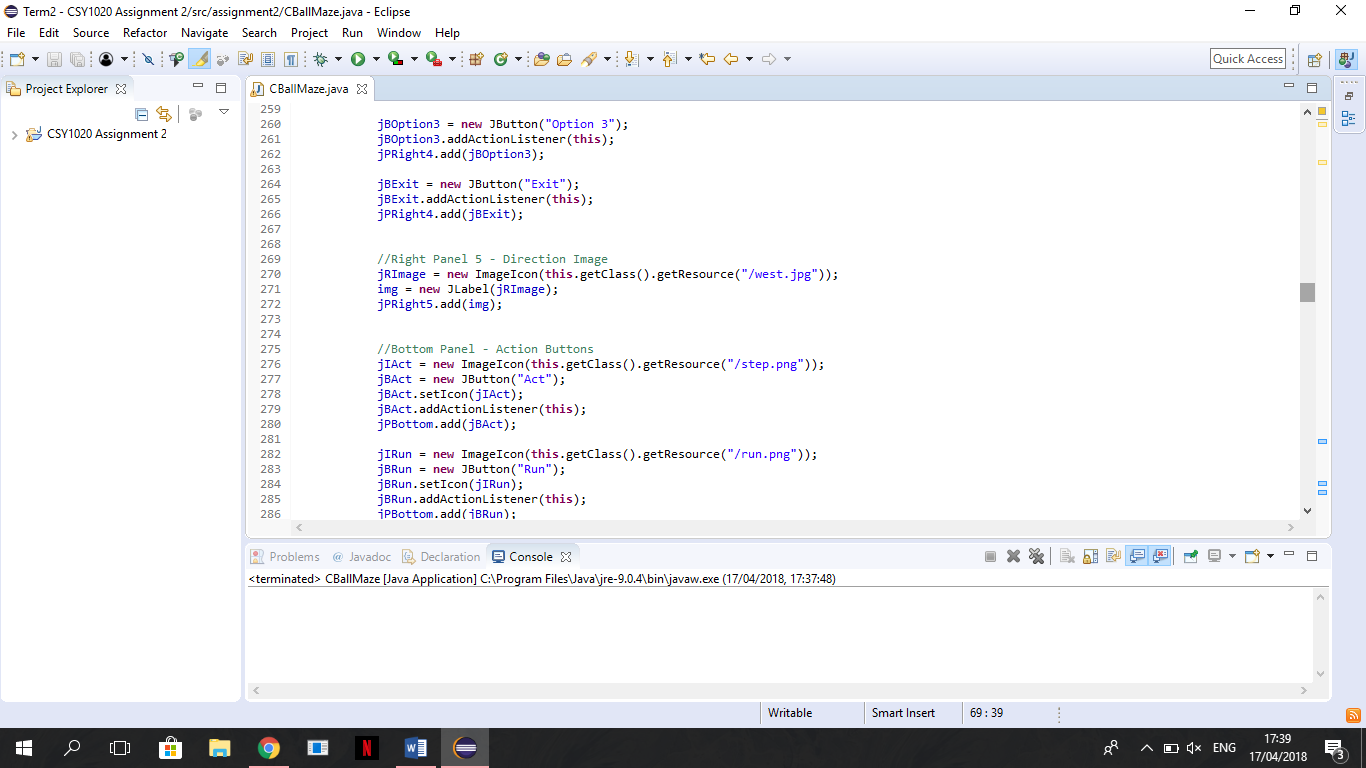
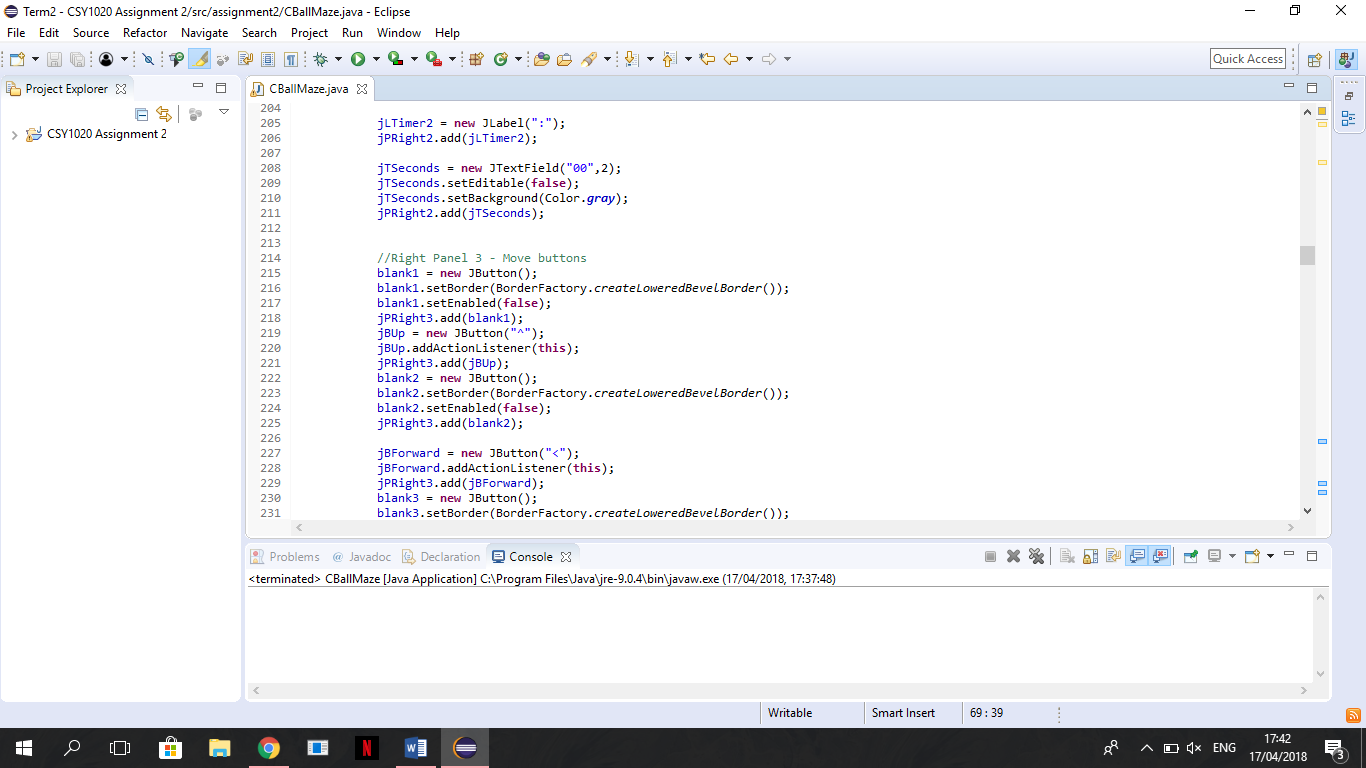
*Figure 6.1: Code result*

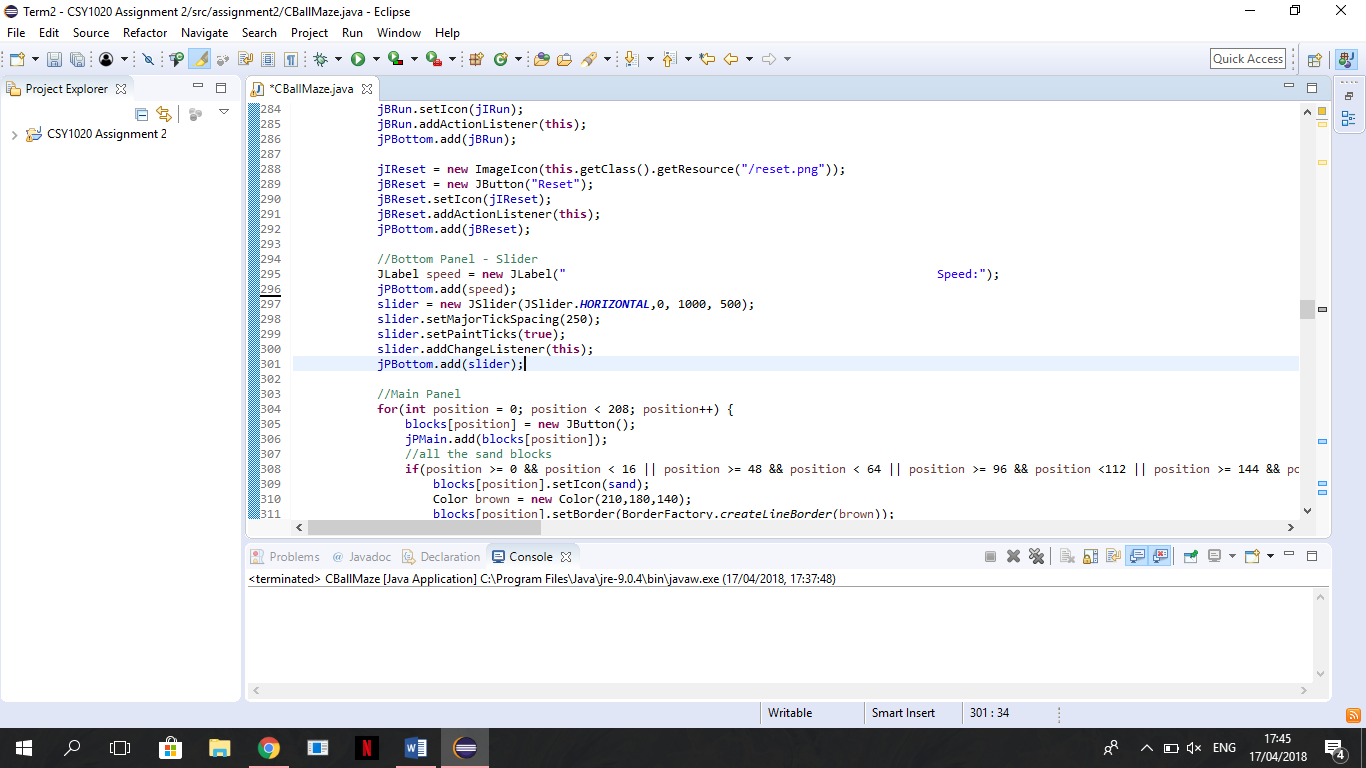
# 

# 

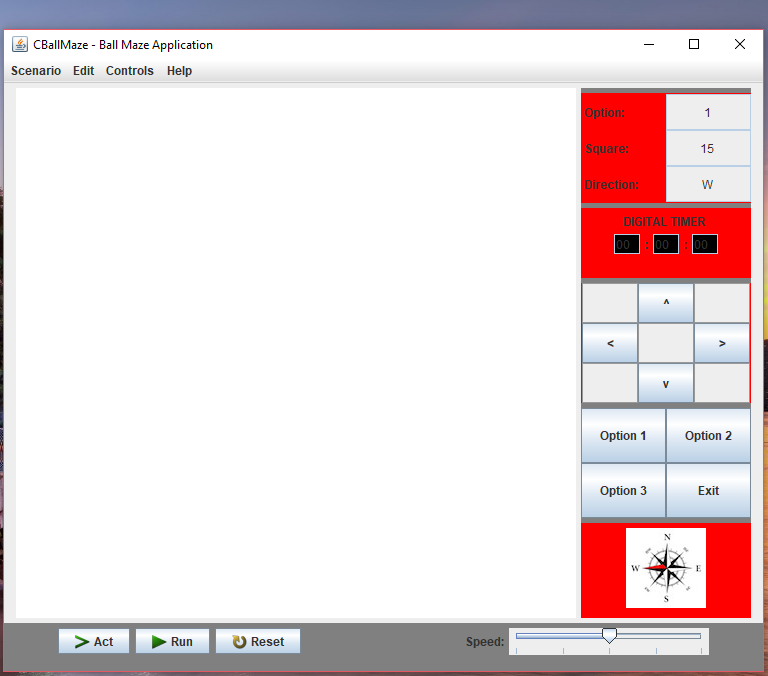
*Figure 6: Main method*

After the frame creation there is the createGUI method, called in the main method (image above line 75), to create the other application features. It’s on this method that all the variables before declared begin to take shape. The code below shows how panels, labels, text fields, buttons, images and sliders were created and the GUI result.



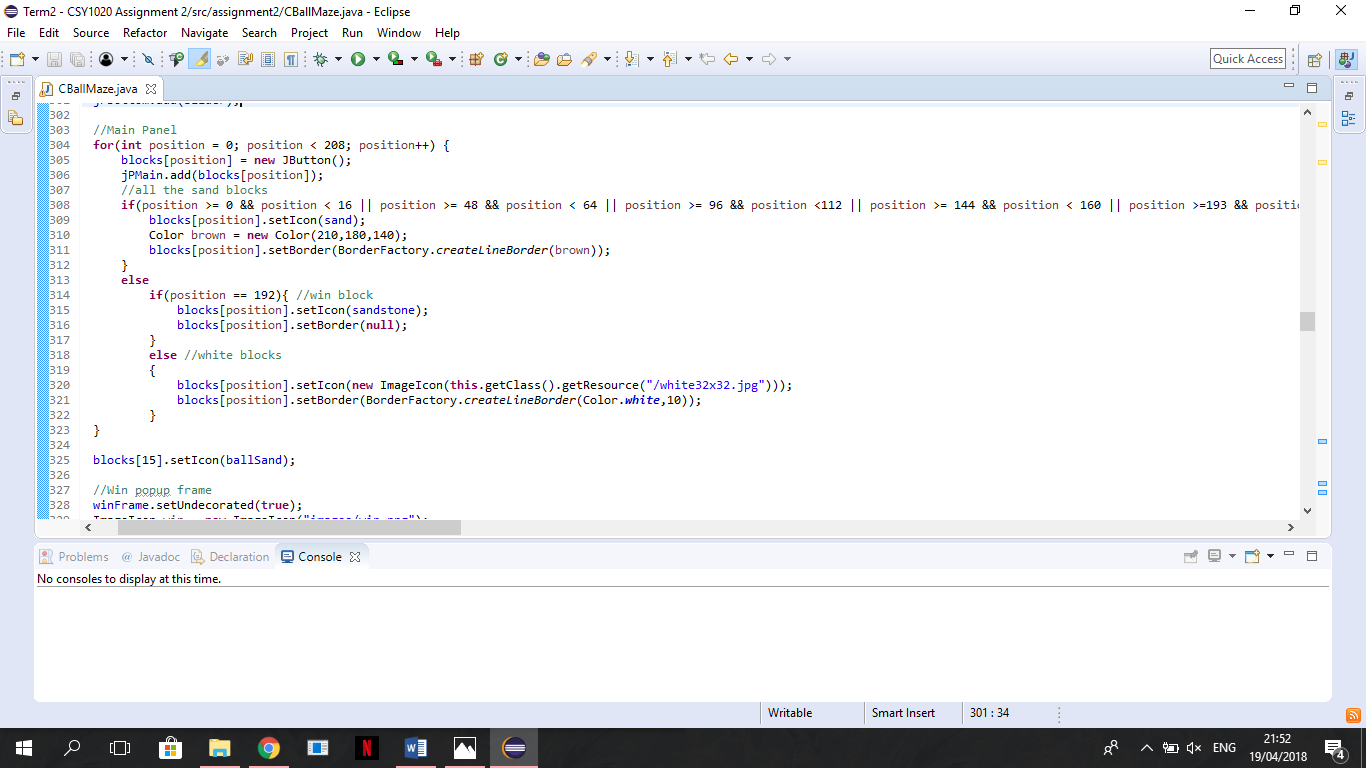


*Figure 7: createGUI() components creation*



*Figure 7.1: Code results*

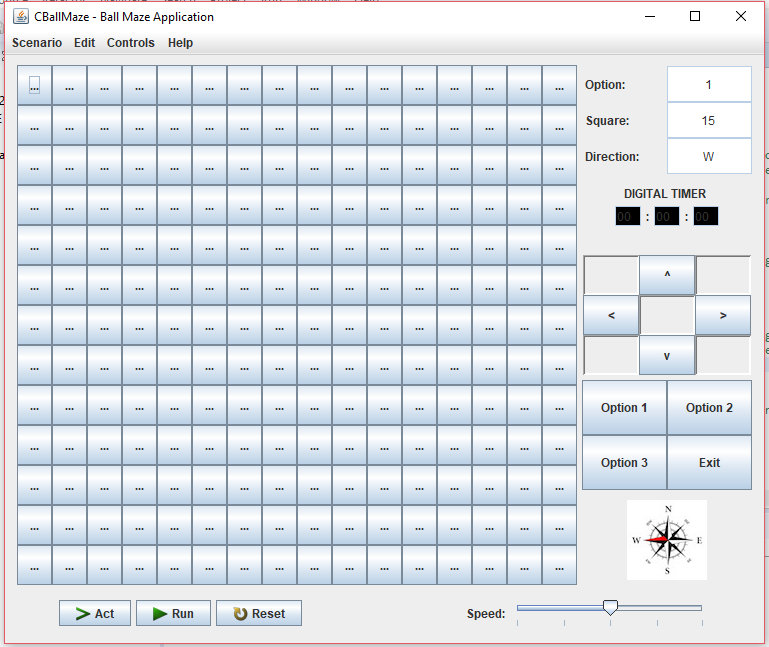
It’s also in the createGUI method that the game model is created, using a for statement. The for statement is used to execute a set of statements repeatedly until a condition is satisfied. In this case, it was used to create 208 buttons for the blocks layout of the game.



*Figure 8: Game interface*

# 

*Figure 9: CBallMaze.java Application – Final result*



*Figure 8.1: Code result*

The image above shows the result of the for statement without the if’s.

The if statements inside the for statement are used to define the buttons style/imagens. Achieving this way, the design asked as you can see in the left image.

# 5. TESTING

|  |  |
| --- | --- |
| System Requirements: Essential (Graphical User Interface): | Visto |
| 13 x 16 grid of **JButton**’s or Icon’s. |  |
| 4 **JButton**’s for the game options ‘Option 1, Option 2, Option 3’ and ‘*Exit*’. | Visto |
| 3 **JButton**’s for ‘Act’, ‘Run’ and ‘*Reset*’. | Visto |
| 9 **JButton**’s for ‘*Forward >*’, ‘Backwards <’, ‘Up ^’, ‘Down v’ should move the ball in the appropriate direction by one square for each press (plus 5 blank). | Visto |
| The compass icon (**JButton)** should illustrate the current direction for the ball. | Visto |
| **3 JLabel**’s for ’Option’, ‘*Square*’ and ‘*Direction*’. | Visto |
| **3 JTextField**’s for the current ‘Option’, Location/*’Square’* and *‘Direction’* of the ball. Use the square identification method e.g. 0 to 207 and N, E etc. | Visto |
| 3 **JLabel’s** for the ‘DIGITAL TIMER and the two ‘:’, with 3 **JTextField’**s for the hours, minutes and seconds. | Visto |
| Create a **JFrame** application, which opens to the set size (775 \* 650). | Visto |
| **JFrame** title set as "*CBallMaze – Ball Maze Application*". | Visto |
| System Requirements: Additional (Functionality & Complexity): |  |
| Application icon for the **JFrame** used (Windows only). | Visto |
| Application dock icon. | Visto |
| The ‘Run’ **JButton** should show the ball moving between the continuously from the initial position (Option 1 – default opening state – ball top right-hand corner) to the end position at the grey square/tile (bottom left-hand corner. | Visto |
| The ‘Reset’ **JButton** should clear/reset the application to its starting/default opening state. | Visto |
| The ‘Act’ **JButton** should step through the above ‘Run’ sequence one move at a time. | Visto |
| Discuss and implement the different options for the 3 configurations. |  |
| The ‘Option 1, Option 2, Option 3’ **JButton**’s should display different tile/object configurations/locations. |  |
| A **JMenuBar** could be included with **JMenu**’s for the *Scenario, Edit, Controls* and *Help*, which include **JMenuItem**’s of *Exit (Scenario)*, *Help Topic* and *About (Help)*. | Visto |
| Additional **JButton**’s may be used to improve the applications usability e.g. ball movement – in random/predefined direction, jump objects/obstacles in Option 2 or 3 etc. |  |
| The ball drops down the maze. | Visto |
| A sound effect is heard when the ball drops down to the next level. | Visto |
| Create a **JFrame** application, which is not resizable. | Visto |
| Create a **JFrame** application, which centres itself on the monitor. | Visto |
| Discuss the possibilities for incorporating intelligence/checks for whether moves are valid. | Visto |
| Digital Timer should start and stop when ‘Run’ is pressed and stopped when a ball gets to the end. | Visto |
| Implement intelligence/checks for whether moves are valid. | Visto |
| **CBallMaze.java & CBallMaze.class** | Visto |
| **Predicted Grade** | **A-/B+** |

Key: Blue GUI; Yellow Testing Application; Red Code.

***Table 1 Testing Criteria***

# 6. CONCLUSION & RECOMMENDATIONS

The CBallMaze application wasn’t a hard thing to do. It’s a simple and easy application to use, all you need to do is click on the directions keys to play or on the run or act button to see the easiest path to the end.

The main objective of the project was to make the application look like the image provided by the lecturer. To make that possible, different features as JPanel’s, JSlider, JButton’s, sounds, Timer, JDialog, JLabel’s and JTextField’s, and an important routine where used. This routine was to create 208 buttons, which are the base of the game. The buttons combinate with the jPMain panel grid layout, create the game interface.

Another important part of the application was the buttons and what they needed to do. Mainly the buttons, on the jPBottom, that allow the user to control the game actions. The act button calls the moveBall method that checks if the ball can move down or should move left; The run start a timer and while the timer is working do clicks on the act button; The reset button set all features of the application to how they were when we opened the application; And the slider allows to change the speed of the ball movements.

Although before the demo class some of the requested features weren’t well done, namely the slide feature and the errors appearing on the console. After looking at the code again and do the necessary changes, I was able to resolve these problems leaving only the option 2 and 3 buttons features undone.

I think the biggest improvement that could be made right now in the application would be move the ball when the keyboard arrow keys are pressed and make the option 2 and 3 buttons work, as it would allow the user to have different interfaces, making the game more interesting.

# REFERENCES

Eclipse Foundation (2017). Eclipse SDK 4.7.3 [online]. Available from: [http://www.eclipse.org](http://www.eclipse.org/) [Accessed 12 Feb 2018]

Java Code Geeks – A simple Timer example. Available from: <https://examples.javacodegeeks.com/desktop-java/awt/event/a-simple-timer-example/> [Accessed 15 March 2018]

JavaTutorials101 - Java: How to play .wav files. Available from:

<https://www.youtube.com/watch?v=QVrxiJyLTqU> [Accessed 04 April 2018]

Bell, D., & Parr, M. (2010). *Java for students*, 6th ed. Upper Saddle River, NJ: Prentice Hall, ISBN 027373122X.

# 