**CSY1020: Problem Solving & Programming**

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

Starting Date: **Friday, 25th May 2018**

Finish Date: **Sunday, 20th August 2018, by 23h59.**

**Please note:** **Viva’s Week commencing: 14thth Augustl 2018 during your normal timetabled session**.

**E-Submission through Turnitin on NILE as TWO separate WORD doc/x.**

**Document 1 = Report & Document 2 = Appendix (Full source code listing).**

**To do this go to the NILE site for this module and under assignments use the link labelled ‘Submit your work’, then the ‘Programming Report’, ‘Appendix’. Please remember this is an individual assignment and all assignments will be submitted through turnitin.**

**Brief:**

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**.

**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)

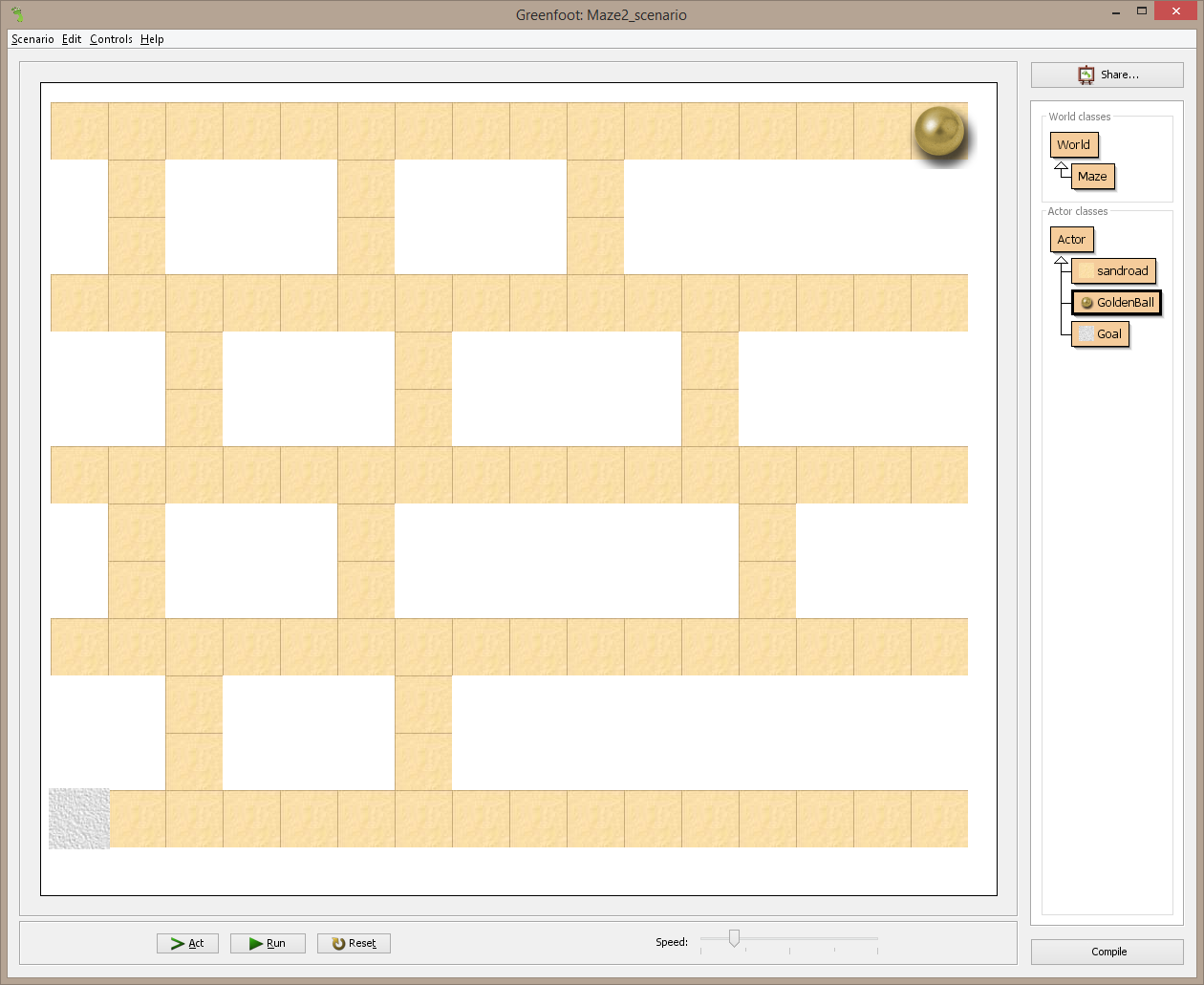
* + For higher grades on the solution part of the assignment see the marking scheme/rubric. You must NOT change the layout and all changes should still meet the criteria of **Rules (Basic**).

It is expected that you may attempt to use and adapt your previous routines produced as part of your CSY1020 Assignment 1.

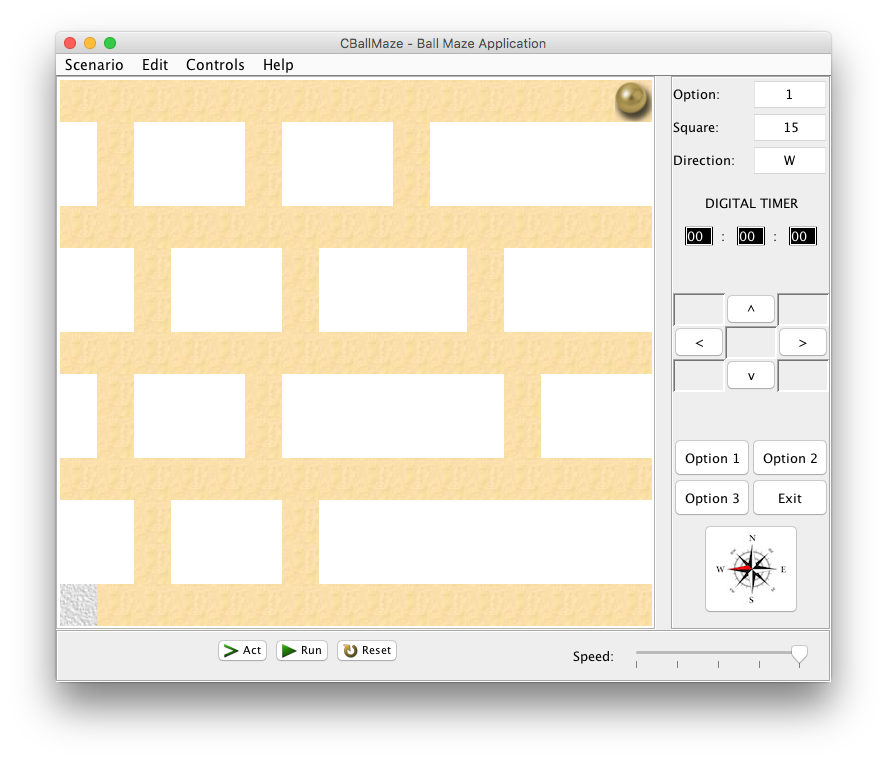
Note:

A word and pdf version of this assignment brief can be found on the Virtual Learning Environment.

Attempt to emulate the following application:



*Figure 1: Problem*



*Figure 2: Emulation - CBallMaze.java Application (Note: Opening State MacOSX).*

**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*".

# 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**

**Deliverables**

**All** requirements (both A & B) **MUST** be delivered to achieve a pass grade for this assignment.

**A) Technical Report**

**Document 1 = Report**

The report should be **2500 words (minimum)** and should be structured as follows:

* Title Page
* Table of contents (to include List of Tables and List of Figures)
* **1. Introduction**/ Problem statement for the task
* **2. Analysis** and assumptions for the task
* **3. Design** for the task in either pseudo-code or flowchart form (include GUI design fully labelled). Include a routine suitable to successfully complete the task.
* **4. Implementation** (Explanation and Morphology of the code and resulting GUI).
* **5. Testing** (A test plan for the task, which should include a call to the test method **moveBall();** This method should solve the room).
* **6. Conclusion & Recommendations** (which should include a critical appraisal of the strengths and weaknesses of your design and what improvements could be made).
* **References** (Using **Harvard** referencing).
* **Bibliography** (If needed, using **Harvard** referencing).

**Document 2 = Appendix**

* Appendices (**The Appendices should include a commented file listing of all source code**).

It is expected that your report should make appropriate use of screen shots, particularly during the Implementation and Testing sections.

## B Viva/demonstration

The viva/demonstration of the developed application will be conducted within the timetabled sessions. Attend your normal timetabled session and the marking tutor will work around the room (You will be free to leave once your software has been demonstrated). The expected viva duration will be between 5 to 10 minutes. Present will be the Module Tutor/Teaching Assistant and the student/s under assessment. *Note: Additionally, students at Edulink, Sri Lanka, Amity, Singapore and DEI, Greece – should submit a 5-10 minute video (voice over/screen capture) of their viva/demonstration. This will be arranged with your module tutor.*

**Useful resources:**

Guide To Writing Technical Reports at :

<http://www.eng.nene.ac.uk/~gary/BScC/reportwriting.html>

One of the past assignment reports at:

<http://194.81.104.27/~gary/csy1020/20042005/csy1020report0405.pdf>.

Rough Guide to Harvard Referencing:

<http://studyskillshub.files.wordpress.com/2013/10/harvardrefquickguide2.pdf>

Screen shot software:

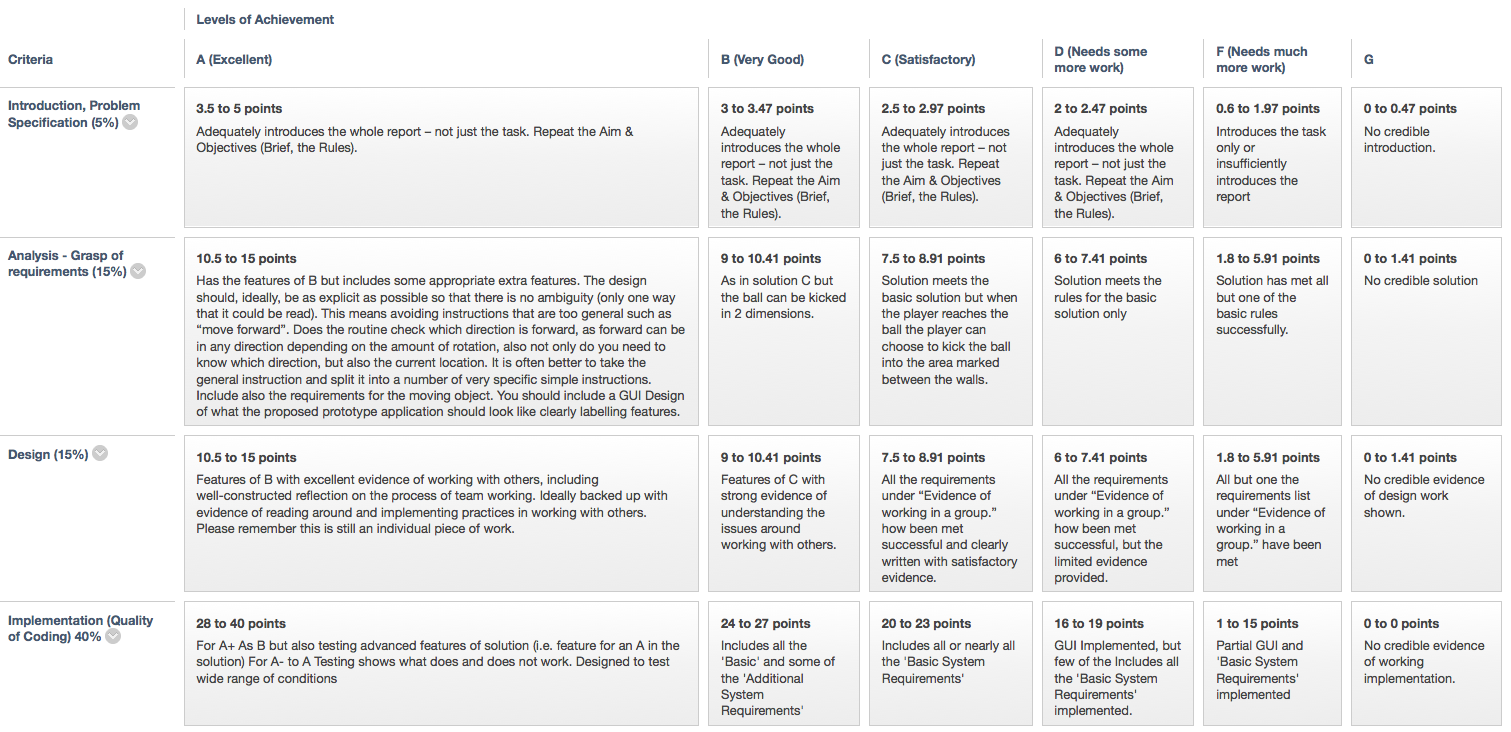
<http://www.mirekw.com/winfreeware/mwsnap.html>

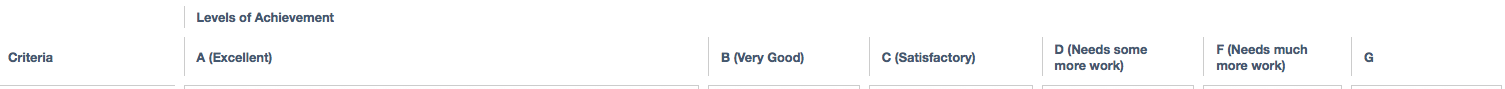
**Learning Objectives.**

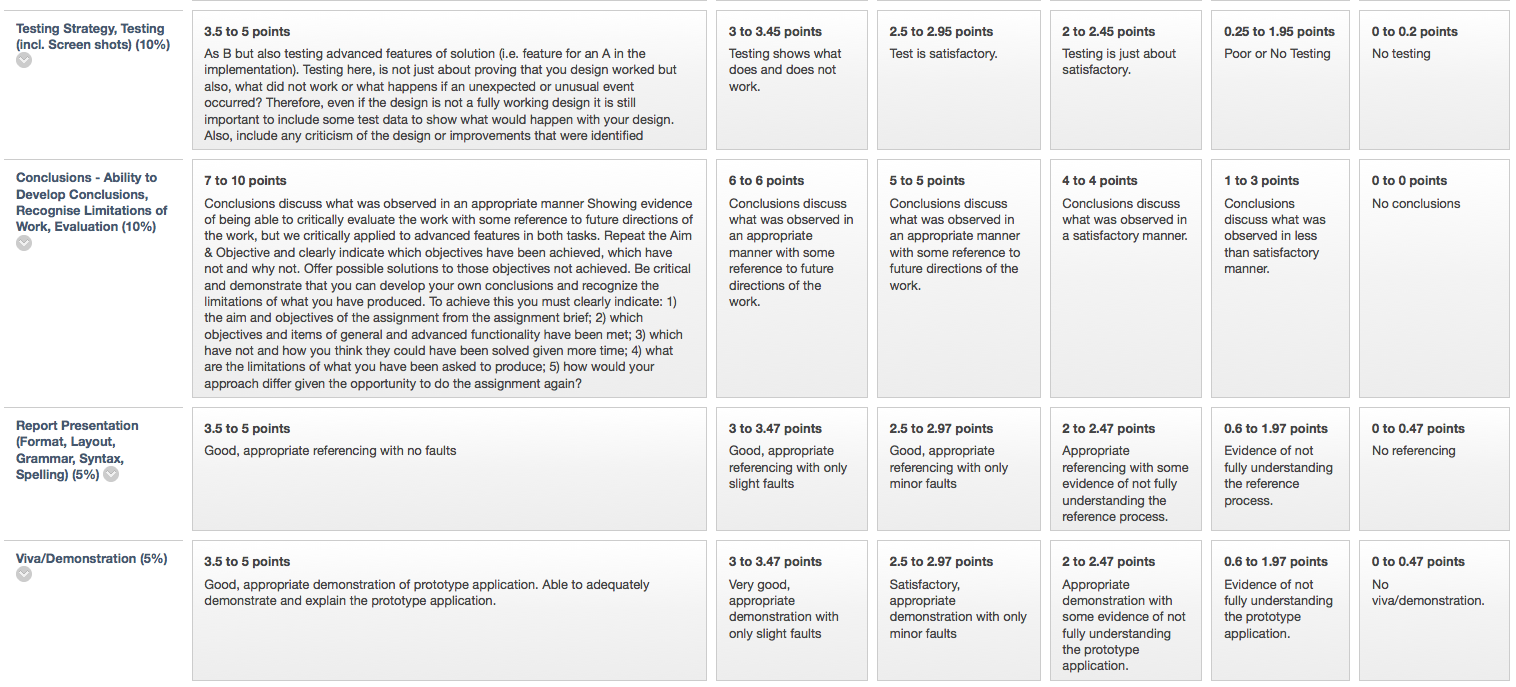
1. Appreciate the principles and practice of analysis and design in the construction of robust, maintainable programs, which satisfy their specifications.
2. Design, write, compile, test and execute straightforward programs using a high level language; appreciate the principles of programming.
3. Appreciate the need for a professional approach to design and the importance of good documentation to the finished programs.
4. Use an appropriate programming language to construct robust, maintainable programs, which satisfy their specifications.
5. Design, write, compile, test and execute programs taking into consideration principles of programming.
6. Apply skills to enable the solution of problems with the construction of appropriate algorithms and a computer program.

Please use the rubric at the end of this document to help you with the assignment and to help with self-feedback.

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem Solving & Programming**  **(Assignment 2: Programming (Java)).** | | | |
| Due for Issue for week commencing: |  | **Date for**  **Submission:** |  |







**Grading Criteria:**

This Standard Front Sheet gives a clear indication of how the grade for this assignment is achieved. In general the following criteria will act as an overall guide to what you should expect:

A **bare pass (D)** will involve incorporate most of the ‘Basic System Requirements:’ above and accompanying technical report covering all appropriate sections.

A **good pass (B to C)** will incorporate all of the ‘Basic System Requirements:’ above and accompanying technical report covering all appropriate sections.

# A very good pass (A) will incorporate most of the ‘Additional System Requirements (functionality & complexity):’ above and accompanying technical report covering all appropriate sections.

## Viva/demonstration

This is a **COMPULSORY** activity, which is considered an essential element of this assignment – you will not pass the assignment **if you don’t attend – a maximum grade of an F** will be given to any assignment submission that has not been demonstrated during the allocated session above.

**Technical Report**

**All** requirements (both A & B) **MUST** be delivered to achieve a pass grade for this assignment. A) Technical Report - Document 1 = Report & Document 2 = Appendix – Code Listing. If the Technical Report is not submitted a G grade will be awarded and if submitted up to 1 week late a maximum of D- will be awarded.

**Notes on achieving the most from this assignment and where marks can be lost:**

1. Carefully read and read again the assignment brief. Following the instructions given carefully and check the front sheet to see where marks are allocated.
2. Always write in the third person i.e. **NO** *I* thought this, *I* did this etc. If you need to refer to yourself use “The author felt” etc.
3. The application **must** be demonstrated. Anyone not demonstrating receives a maximum of F.
4. If an assignment is not submitted (even if the application was demonstrated) a mark of G is given.
5. The brief states that “The source code file containing the **main()** method and the compiled byte code class files should be named as follows: **CBallMaze.java** & **CBallMaze.class**”.
6. A Title Page should be attached to the assignment.
7. The **introduction/problem statement** should repeat the information given in the assignment brief (available in wordIcon word).
8. **Implementation** should be a commentary of the code explaining how the design has been applied. Screen shots should shows snippets of the code and the outcome in the GUI.
9. The FULL source code in the Appendix should have a header indicating the usual information e.g.

/\*\*  
Program: Assignment 2: Application – Ball Maze  
Filename: CBallMaze.java   
@author: © Gary Hill (200WXYZ)   
Course: BEng/BSc/HND Computing Year 1   
Module: CSY1020 Problem Solving & Programming   
Tutor: Gary Hill   
@version: 2.0 Incorporates Artificial Intelligence!   
Date: 23/11/17   
\*/

1. Meaningful variable names should be used in code e.g. jBRotate (for the rotate JButton).
2. Your code should have meaningful comments e.g. //button to move object down/south.
3. The aim of the **Conclusion & Recommendations** section is to demonstrate that you can develop your own conclusions and recognize the limitations of what you have produced. To achieve this you should clearly indicate:

* the aim and objectives of the assignment from the assignment brief;
* which objectives and items of general and advanced functionality have been met;
* which have not and how you think they could have been solved given more time;
* what are the limitations of what you have been asked to produce;
* how would your approach differ given the opportunity to do the assignment again?

1. **Analysis:** It is expected in the analysis that you will identify issues such as for example:
   * What inputs are needed and the form of the inputs?
   * What is the form of the output or outputs?
   * What rules are needed?
2. **Design:** The design should, ideally, be as explicit as possible so that there is no ambiguity (only one way that it could be read). This means avoiding instructions that are too general such as “move forward”. Does the routine check which direction is forward, as forward can be in any direction depending on the amount of rotation, also not only do you need to know which direction, but also the current location. It is often better to take the general instruction and split it into a number of very specific simple instructions. Include also the requirements for the robot.
3. **Implementation:** The implementation should be a commentary of the code explaining how the design has been applied. Screen shots should shows snippets of the code and the outcome in the GUI, these should also demonstrate/illustrate key areas of the code e.g. nested for loop. Quality of coding should also be considered e.g. use of meaningful variables, meaningful comments, neatness/readability, attempt to use a coding convention, header/title etc.
4. **Testing:** Testing here, is **not just about proving that you design worked but also, what did not work or what happens if an unexpected or unusual event occurred?** Therefore, even if the design is not a fully working design it is still important to include some test data to show what would happen with your design. Also, include any criticism of the design or **improvements** that were identified.

**Remember to Reference your Software Code**

Author (Year) *Title of Program* (Version Number) [format type] (computer program, software or code, Place of publication: publisher (if available). Available from: URL (if online).

**E.g.**

In Text or Code: (Eclipse, 2017)

In Reference list:

Eclipse (2014) Eclipse Lunar (Version 4.4.1) [Software] Eclipse.org. Available from: <http://www.eclipse.org/downloads/> [Accessed 12 Feb 2014]

**1 Coding Conventions**

A set of coding conventions should be followed throughout, which help in the creation of code which is more efficient, maintainable, robust and most importantly – readable. The following guidelines should be used:

**Naming Rules:**

Variables were prefixed with lowercase mnemonic indicating nature of object. E.g. jBVariable(JButton), iconCar(ImageIcon), fVariable(float), dVariable(double), boxVariable(Box), bVariable (boolean), jTHello (JTextfield), jLDirection (JLabel), nSquare (int) bgVariable (BranchGroup) etc.

**Align brackets:**

By default eclipse places starting bracket for a code construct at end of the starting line, and aligns the closing bracket with the first column of the construct. This was changed to align the starting and ending brackets on the same columns, so that a reader can clearly identify the borders for a code construct. E.g.:

        for (int i=0; i<20; i++)

        {

                // … Place code here

}

**Code Indentation:**

A practice often ignored by most programmers, indenting is a good practice that helps immensely with code readability and debugging.

**Commenting:**

All important areas of the code should be commented, clearly explaining the functionality. As with some of the techniques above, this helps with code readability, maintenance and debugging.

**2 References**

Eclipse Foundation (2017). Eclipse SDK 4.4.1 [online]. Available from: [http://www.eclipse.org](http://www.eclipse.org/) [Accessed 12 Feb 2017]

ORACLE (2017a) Lesson: Exceptions. Available from: <http://java.sun.com/docs/books/tutorial/essential/exceptions> [Accessed 20 may 2014].

Wikipedia (2017) Code Refactoring [online]. Available from: <http://en.wikipedia.org/wiki/Refactoring> [Accessed 12 May 2017].