

CSY1020: Problem Solving & Programming
Assignment 2: Programming (Java) (50%)Starting Date: **Week beginning 25th November 2019**Finish Date: **Sunday, 3rd May 2020, by 23h59.****Please note: Viva's Week commencing: 27th April 2020 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 ball being kicked (navigated) around the pitch 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 can move anywhere within the pitch and across the wall in the middle of the screen.
- If the ball touches a baby it is deflected/rebounds back in the opposite direction.
- The ball must move one whole 'white/wall' block at a time every time a movement button (via a direction button (<, > v ^)) /key is pressed (when movement is possible).
- The solution must use the scenario provided. i.e. If the babies are left in the unmodified codes starting position the ball would move between them continuously.
- The ball must stop when the perimeter of the pitch is reached.
- The basic solution must be completed using the 'act' button (accessing the **kickBall()** method within the **CBabyBallBounce.class**).

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

- Your solution must still use the scenario provided (all the basic features above).
- Add appropriate extra features to the solutions, e.g. a) The ball can bounce off a baby or babies in random direction, b) Two new 'player' babies are added to each side of the line, that move vertically towards the ball ready to potentially bounce the ball back, c) Add a scoring system for each side (a, b or c for **Intermediate**, a, b & c for **Advanced**).
- For higher grades on the solution part of the assignment see the marking scheme at the front of the brief.
- 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.

Attempt to emulate the following application:

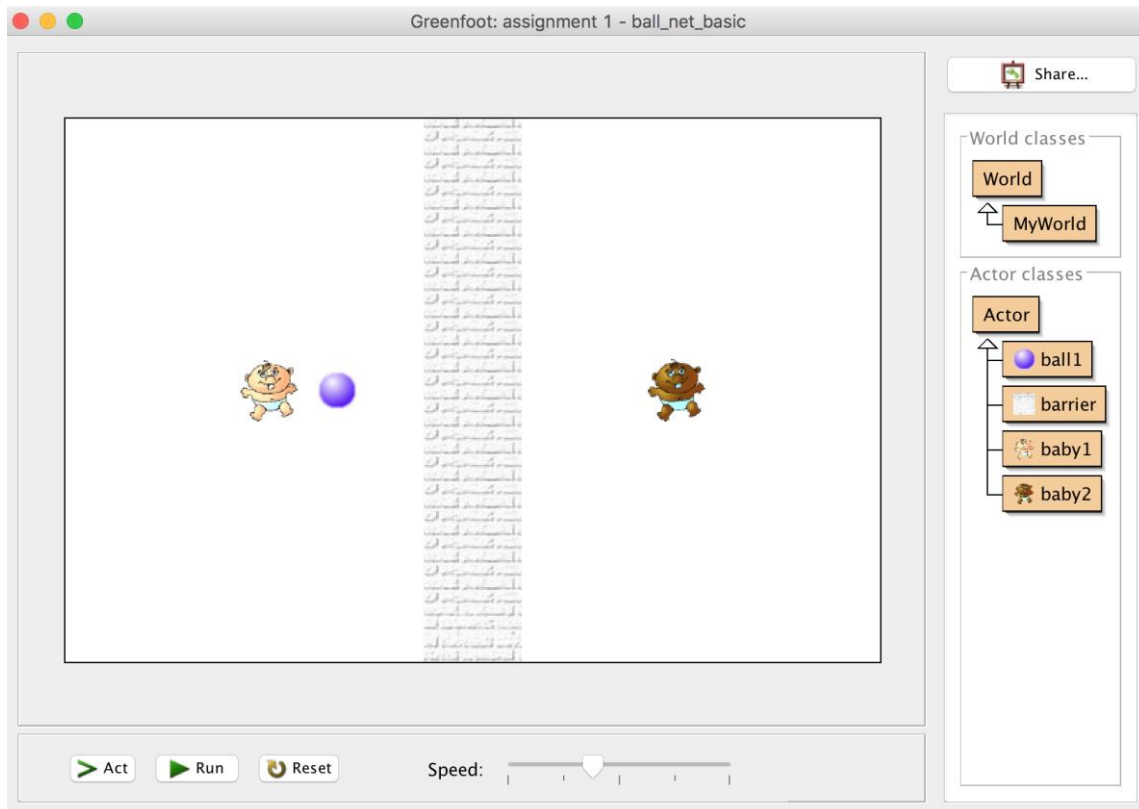


Figure 1: Problem

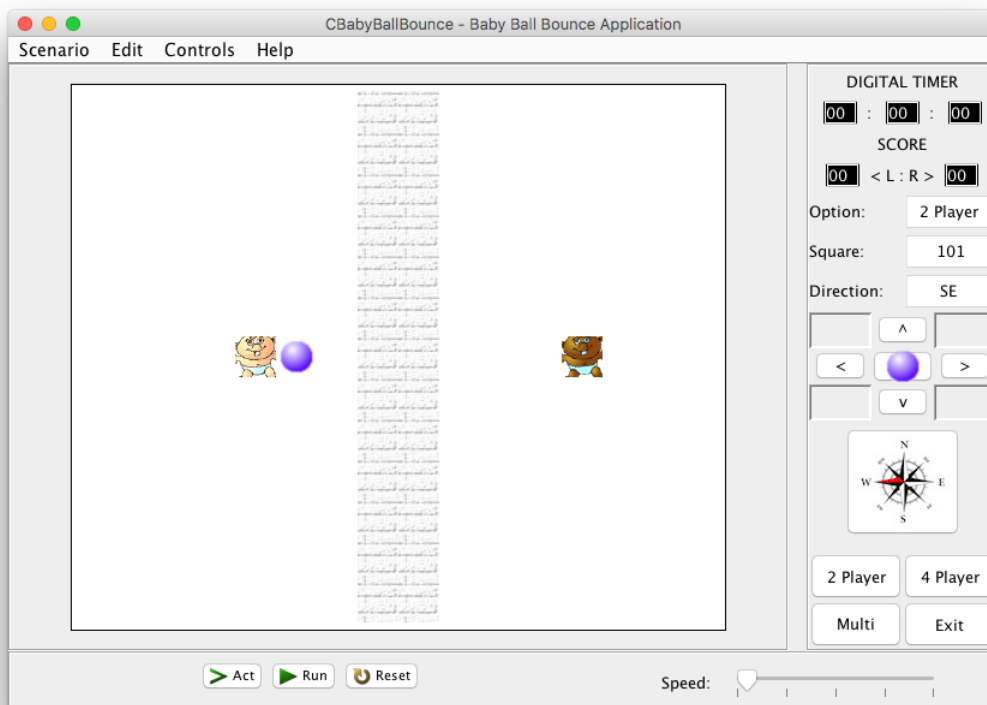


Figure 2: Emulation - CBabyBallBounce.java Application – Opening State Win10.

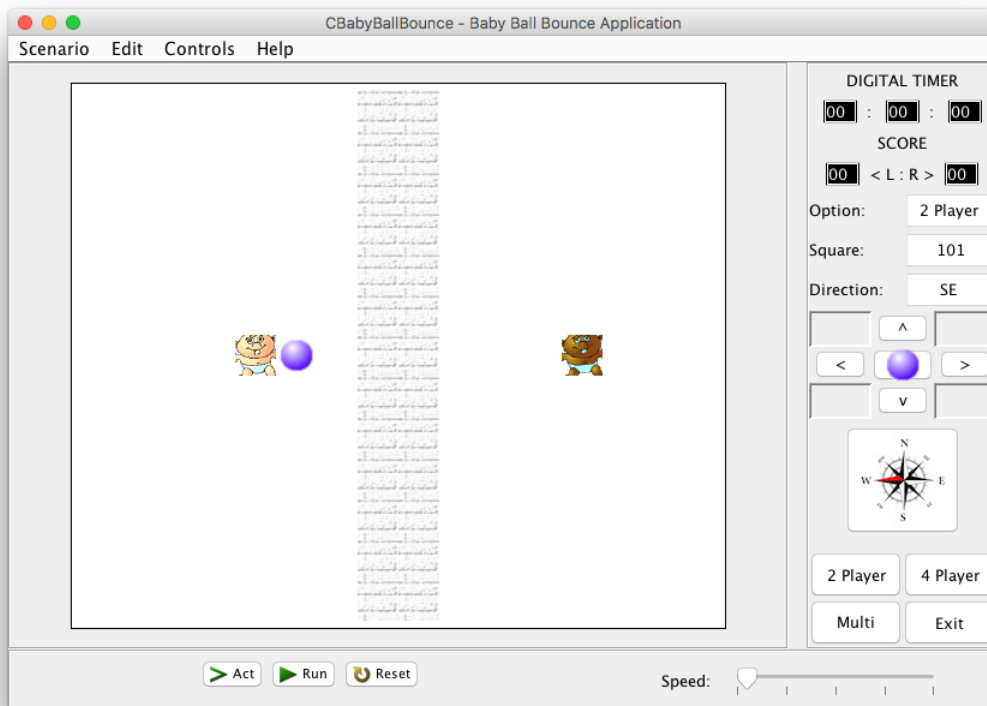


Figure 3: Emulation - CBabyBallBounce.java Application – Opening State MacOSX.

The images for the system that can be seen in the snapshots (such as the babies and ball) can be downloaded from the assignment folder on NILE

System Requirements

Essential (Graphical User Interface):

- 13 x 16 grid of **JButton**'s or Icon's.
- 4 **JButton**'s for the game options '2 Player, 4 Player, Multi' 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 **TextField**'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 **TextField**'s for the hours, minutes and seconds.
- 2 **JLabel**'s for the 'SCORE and '<L:R>', with 2 **TextField**'s for the scores (L & R).
- Create a **JFrame** application, which opens to the set size (825 * 585).
- **JFrame** title set as "CBabyBallBounce – Baby Ball Bounce Application".

Additional (Functionality & Complexity):

- Application icon for the **JFrame** used.
- The 'Run' **JButton** should show the ball moving between the babies continuously from the initial position (2 Player – default opening state).
- 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 '2 Player, 4 Player, Multi' **JButton**'s should display different obstacle/car 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 bounce – in random direction, deflection angle etc.
- Create a **JFrame** application, which is not resizable.
- Create a **JFrame** application, which centres itself on the monitor.
- Use of additional baby images indicating the current position and direction of the baby.
- 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 baby misses the ball (with the ball continuing to the left or right boundary and stopping itself and the timer).
- Implement intelligence/checks for whether moves are valid.
- A **kickBall()** method should be used to solve the problem. The **kickBall()** method should include **move(left), move(right), move(up), move(down)** methods (see below).

```
public void kickBall()
{
    move(left);
    .....
    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:

CBabyBallBounce.java & CBabyBallBounce.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 **kickBall()**; 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 for **Week 30**. 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.

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.

- a) Appreciate the principles and practice of analysis and design in the construction of robust, maintainable programs, which satisfy their specifications.
- b) Design, write, compile, test and execute straightforward programs using a high level language; appreciate the principles of programming.
- c) Appreciate the need for a professional approach to design and the importance of good documentation to the finished programs.
- d) Use an appropriate programming language to construct robust, maintainable programs, which satisfy their specifications.
- e) Design, write, compile, test and execute programs taking into consideration principles of programming.
- f) 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.

Personal Development & Key Skills (for your PDP)

This assignment provides an opportunity to add to your personal development portfolio as indicated below:

PDP: Personal development elements (for your PDP), this assignment provides an opportunity to add to your personal development portfolio the following:

- Problem-solving – the whole assignment is concerned with this area;
- Use of number – tasks involve manipulating numbers to solve a problem;
- Managing the Learning process – Successfully completing the assignment on time.
- Communication skills – presenting your work and critically appraising it in a clear form.

Key Skills	Y/N
1 Managing the Learning Process: Ability to evaluate learning styles, identify strategies for approaching study tasks, manage and organise oneself taking responsibility for decision-making, target setting and delivery of action.	Y
2 Communication Skills: The ability to express, discuss and present knowledge, ideas and viewpoints to a variety of audiences with confidence and clarity.	Y
3 Groupwork: The ability to work harmoniously and productively as a member of a group in a variety of roles, demonstrating an awareness of group dynamics, appropriate inter personal and interactional skills.	N
4 Information Skills: The ability to identify information needs, access and evaluate a range of relevant sources, organise and use information efficiently and effectively for both academic and professional purposes.	Y
5 Problem Solving: The ability to identify problems and to apply concepts, principles and techniques in order to generate solutions, choose between alternatives and take appropriate action.	Y
6 Use of IT: The ability to effectively use key information technology and appropriate software to assist in the learning process through research and retrieval, communication and manipulation of information in various forms.	Y
7 Application of number: The ability to understand, interpret and use numerical and graphical information accurately and effectively.	Y

Remember to consult and completing your **Key Skills Checklist** or **Skills Development Plan** as appropriate, to help you to identify current strengths and how you can build on these, as well as highlight areas that need improving. At the end of each year you can complete a new skills checklist/plan, based on your learning experiences.

Year 1:

http://pdp.northampton.ac.uk/Year1_Files/docs/Key%20Skills%20Checklist.doc





Year 2:

http://pdp.northampton.ac.uk/Year2_Files/Docs/Stage2SkillsDevelopmentPlan.doc

Year 3

http://pdp.northampton.ac.uk/Year3_Files/Docs/Skill%20Development%20Plan%20year%203%20generic.doc

Problem Solving & Programming						
(Assignment 2: Programming (Java)).						
Due for Issue for week commencing:	Monday, 25 th October 2019	Date for Submission:	Sunday, 3 rd may 2020 23h59 via NILE			
	Levels of Achievement					
Criteria	A (Excellent)	B (Very Good)	C (Satisfactory)	D (Needs some more work)	F (Needs much more work)	G
Introduction, Problem Specification (5%) ⌵	3.5 to 5 points Adequately introduces the whole report – not just the task. Repeat the Aim & Objectives (Brief, the Rules).	3 to 3.47 points Adequately introduces the whole report – not just the task. Repeat the Aim & Objectives (Brief, the Rules).	2.5 to 2.97 points Adequately introduces the whole report – not just the task. Repeat the Aim & Objectives (Brief, the Rules).	2 to 2.47 points Adequately introduces the whole report – not just the task. Repeat the Aim & Objectives (Brief, the Rules).	0.6 to 1.97 points Introduces the task only or insufficiently introduces the report	0 to 0.47 points No credible introduction.
Analysis - Grasp of requirements (15%) ⌵	10.5 to 15 points Has the features of B but includes some appropriate extra features. 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 moving object. You should include a GUI Design of what the proposed prototype application should look like clearly labelling features.	9 to 10.41 points As in solution C but the ball can be kicked in 2 dimensions.	7.5 to 8.91 points Solution meets the basic solution but when the player reaches the ball the player can choose to kick the ball into the area marked between the walls.	6 to 7.41 points Solution meets the rules for the basic solution only	1.8 to 5.91 points Solution has met all but one of the basic rules successfully.	0 to 1.41 points No credible solution
Design (15%) ⌵	10.5 to 15 points Features of B with excellent evidence of working with others, including well-constructed reflection on the process of team working. Ideally backed up with evidence of reading around and implementing practices in working with others. Please remember this is still an individual piece of work.	9 to 10.41 points Features of C with strong evidence of understanding the issues around working with others.	7.5 to 8.91 points All the requirements under “Evidence of working in a group.” how been met successful and clearly written with satisfactory evidence.	6 to 7.41 points All the requirements under “Evidence of working in a group.” how been met successful, but the limited evidence provided.	1.8 to 5.91 points All but one the requirements list under “Evidence of working in a group.” have been met	0 to 1.41 points No credible evidence of design work shown.
Implementation (Quality of Coding) 40% ⌵	28 to 40 points For A+ As B but also testing advanced features of solution (i.e. feature for an A in the solution) For A- to A Testing shows what does and does not work. Designed to test wide range of conditions	24 to 27 points Includes all the 'Basic' and some of the 'Additional System Requirements'	20 to 23 points Includes all or nearly all the 'Basic System Requirements'	16 to 19 points GUI Implemented, but few of the Includes all the 'Basic System Requirements' implemented.	1 to 15 points Partial GUI and 'Basic System Requirements' implemented	0 to 0 points No credible evidence of working implementation.
	Levels of Achievement					
Criteria	A (Excellent)	B (Very Good)	C (Satisfactory)	D (Needs some more work)	F (Needs much more work)	G

Testing Strategy, Testing (incl. Screen shots) (10%) 	3.5 to 5 points <p>As B but also testing advanced features of solution (i.e. feature for an A in the implementation). 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</p>	3 to 3.45 points <p>Testing shows what does and does not work.</p>	2.5 to 2.95 points <p>Test is satisfactory.</p>	2 to 2.45 points <p>Testing is just about satisfactory.</p>	0.25 to 1.95 points <p>Poor or No Testing</p>	0 to 0.2 points <p>No testing</p>
Conclusions - Ability to Develop Conclusions, Recognise Limitations of Work, Evaluation (10%) 	7 to 10 points <p>Conclusions discuss what was observed in an appropriate manner Showing evidence of being able to critically evaluate the work with some reference to future directions of the work, but we critically applied to advanced features in both tasks. Repeat the Aim & Objective and clearly indicate which objectives have been achieved, which have not and why not. Offer possible solutions to those objectives not achieved. Be critical and demonstrate that you can develop your own conclusions and recognize the limitations of what you have produced. To achieve this you must clearly indicate: 1) the aim and objectives of the assignment from the assignment brief; 2) which objectives and items of general and advanced functionality have been met; 3) which have not and how you think they could have been solved given more time; 4) what are the limitations of what you have been asked to produce; 5) how would your approach differ given the opportunity to do the assignment again?</p>	6 to 6 points <p>Conclusions discuss what was observed in an appropriate manner with some reference to future directions of the work.</p>	5 to 5 points <p>Conclusions discuss what was observed in an appropriate manner with some reference to future directions of the work.</p>	4 to 4 points <p>Conclusions discuss what was observed in a satisfactory manner.</p>	1 to 3 points <p>Conclusions discuss what was observed in less than satisfactory manner.</p>	0 to 0 points <p>No conclusions</p>
Report Presentation (Format, Layout, Grammar, Syntax, Spelling) (5%) 	3.5 to 5 points <p>Good, appropriate referencing with no faults</p>	3 to 3.47 points <p>Good, appropriate referencing with only slight faults</p>	2.5 to 2.97 points <p>Good, appropriate referencing with only minor faults</p>	2 to 2.47 points <p>Appropriate referencing with some evidence of not fully understanding the reference process.</p>	0.6 to 1.97 points <p>Evidence of not fully understanding the reference process.</p>	0 to 0.47 points <p>No referencing</p>
Viva/Demonstration (5%) 	3.5 to 5 points <p>Good, appropriate demonstration of prototype application. Able to adequately demonstrate and explain the prototype application.</p>	3 to 3.47 points <p>Very good, appropriate demonstration with only slight faults</p>	2.5 to 2.97 points <p>Satisfactory, appropriate demonstration with only minor faults</p>	2 to 2.47 points <p>Appropriate demonstration with some evidence of not fully understanding the prototype application.</p>	0.6 to 1.97 points <p>Evidence of not fully understanding the prototype application.</p>	0 to 0.47 points <p>No viva/demonstration.</p>

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: **CBabyBallBounce.java** & **CBabyBallBounce.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  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 - Baby Ball Bounce
Filename:   CBabyBallBounce.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:       23/11/16
*/
```

- 10 Meaningful variable names should be used in code e.g. jBRotate (for the rotate JButton).
- 11 Your code should have meaningful comments e.g. `//button to move object down/south`.
- 12 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?
- 13 **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?
- 14 **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.
- 15 **Implementation:** The implementation should be a commentary of the code explaining how the design has been applied. Screen shots should show 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.
- 16 **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, 2014)

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 (2014). Eclipse SDK 4.4.1 [online]. Available from: <http://www.eclipse.org> [Accessed 12 Feb 2014]

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

Wikipedia (2014) Code Refactoring [online]. Available from: <http://en.wikipedia.org/wiki/Refactoring> [Accessed 12 May 2014].

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

- 17 Carefully read and read again the assignment brief. Following the instructions given carefully and check the front sheet to see where marks are allocated.
- 18 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.
- 19 The application **must** be demonstrated. Anyone not demonstrating receives a maximum of F.
- 20 If an assignment is not submitted (even if the application was demonstrated) a mark of G is given.
- 21 The brief states that "The source code file containing the **main()** method and the compiled byte code class files should be named as follows: **CBabyBallBounce.java** & **CBabyBallBounce.class**".
- 22 A "Std. Front Sheet (Attached to this assignment)" should be attached to the assignment with a Title Page.
- 23 The **introduction/problem statement** should repeat the information given in the assignment brief (available in  word).
- 24 **Implementation** should be a commentary of the code explaining how the design has been applied. Screen shots should show snippets of the code and the outcome in the GUI.
- 25 The FULL source code in the Appendix should have a header indicating the usual information e.g.
- ```
/**
Program: Assignment 2: Application - Baby Ball Bounce
Filename: CBabyBallBounce.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: 23/11/16
*/
```
- 26 Meaningful variable names should be used in code e.g. jBRotate (for the rotate JButton).
- 27 Your code should have meaningful comments e.g. `//button to move object down/south`.
- 28 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?
- 29 **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?
- 30 **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.
- 31 **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.
- 32 **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, 2014)

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 (2014). Eclipse SDK 4.4.1 [online]. Available from: <http://www.eclipse.org> [Accessed 12 Feb 2014]

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

Wikipedia (2014) Code Refactoring [online]. Available from: <http://en.wikipedia.org/wiki/Refactoring> [Accessed 12 May 2014]