ROBOT WORLD SIMULATOR FOR JAVA

By

Rogers Bhalalusesa

A dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science of the University of Kent

University Of Kent, Uk

September, 2009

DECLARATION

AND

COPYRIGHT

I, Rogers Bhalalusesa, declare that this dissertation is my own original work and that it has not been presented and will not be presented to any other university for a similar or any other degree award.

Signature …….

This dissertation is copyright material protected under the Berne Convention, the copyright Act of 1999 and other International and national enactment, in that behalf, on intellectual property. It may not be reproduced by any means, in full or in part, except for short extracts in fair dealing, for research or private study, critical scholarly review or discourse with an acknowledgement, without permission of the Directorate of Postgraduate Studies on behalf of both the author and University of Kent.

Acknowledgements

It is a pleasure to thank those who made this project successful First and foremost I would like to take this opportunity to thank my supervisor, Dr Peter Kenny for all his guidance and for support he has given me throughout the project

Secondly I would like to thank my brother, Moses Bhalalusesa who has been living with me throughout the time that I have been busy working on this project.

Additionally I would like to thank all those who participated in the evaluation of the projects for their useful ideas which have contributed so much to the project.

Last but not least I would like to thank my parents for being able to sponsor me throughout the pursue of my master"s degree. There is a lot that they have sacrificed for me to get me to where I am, and for this I will always be proud to have them as my parents

Abstract

Teaching Introduction to Object Oriented Programming and Java by Visualization is widely being adopted by many institutions. Regardless of the nature of the introduction technique used in java, at later stages, the students will have to develop programs by using editors where they will have to manually write codes. The transition from using Java by visualization to that by directly coding is hard. The jump is so big that when the students have learned Java by visualization they still find it hard to write java code for their own programs. Thus a Mid-point is vital to help the students as they move from Java by visualization to Java Coding.

This project introduces a code writer to the java visualization teaching tool known as Robot World Simulator. The code writer will help novice programmers to transition from the learning Java through visualization to java code writing gradually. The Code writer is an interface where students can write Java codes to control the movements of the Robot in the Robot World Simulator and visualize the effects of the code they write without waiting for the program to compile. Students start learning OO concept through visualization and practise code writing by controlling the objects using both code writing and visualization controls before they switch to writing codes completely. The code writer will not only provide assistance when the novice programmers venture into poor programming and make errors but also it will be used side to side with the visualization controls and in that way it will make it possible for the student to transition gradually from visualization to code writing.

The work describes the background of the problem of transition from visualization to code writing, before giving an insight on how the development of the Robot World Simulator with code writer was achieved and shows how the code writer can help in transitioning the students from programming by visualization to programming by coding.

Table of Contents.

Page

List of Tables

Page

1. Introduction.

1.1. Background to the Problem.

Teaching programming is still a big problem in the Education of computer science students. Novice programmers tend to perform poorly in this field of programming. Kaasboll reports that drop-out or failure rates vary from 25 to 80 % world-wide [1]. Most educational institutions introduce students to the concepts of Programming using Object Orientation (OO) approach in Java [2]. Even with the simplicity of Object orientation programming the novice programmers still face the problem of developing to competent programmers.

This can be looked at from the point of view of how the novice programmers are introduced to the concepts of programming. When the novice programmers first start to learn how to program they are faced with a number of difficulties. The students not only have to learn how the Object Orientation is achieved in real word but also they have to learn how to apply it using the Java programming language which has its own syntax that is new to them.

A lot is still being done to help teach students programming language in order to address this problem. Of many approaches that have been taken, Visualization has so far been successful in trying to introduce novice programmers to understand the concepts of programming [3]. However the power of visualization teaching tools limits the students" ability to manipulate systems through visualization controls buttons and through input parameters. This does not allow students to develop programs of their own completely in visualization. Therefore in order to create their own programs the students turn to using text editors that can be

provided by some of these visualization tools and complex Java Integrated Development Environment (IDE) systems that have many advanced features that may overwhelm students with no programming background [4]. The novice programmers hence find it hard to start using the text editors to write java programs after they have learned the concepts of OO through visualization.

1.2. Problem Statement

In view of the facts above the main problem can be summed up in one sentence as "Migration from using visualization controls in the Visualization tools to writing complete java programs in text editors and java IDE's in order to create programs is a big transition for novice programmers". This transition is so big such that most students tend to fails to write programs in the IDE's after they have clearly understood the OO concepts using Visualization.

Robot World Simulator is a tool that uses visualization to introduce Novice programmers to the concepts of OO programming [5]. It has been developed for almost over ten years now at the University of Kent and each one is an enhancement of the previous one. But like many of the java visualization tools it still leaves a big gap when a student wants to migrate from learning by visualization to the actual coding.

1.3. Solution Domain

The solution to the swift transition of programming by visualization to code writing will be based on the principles of interaction between the students and visualization teaching tool.

In order to reduce the gap from Visualization to coding, a midpoint should be established that allows students to interact with the systems by starting to write simple java statements slowly and while at the same time continue to interact with the visualization using visualization control buttons. On top of that the visualization teaching tool should provide multiple views in order to make the student associate the java codes they write with other things they are familiar of like normal English language and Visualization.

Therefore the aim of this project is to develop a Robot World Simulator tool that has all the necessary features of a good visualization teaching tool to help transition the students from learning OO programming by visualization to writing their own codes.

2. Literature Review

2.1. Intergating Interaction to Visualization and Coding

When the students first learn to program by visualization they are allowed to interact with the visualization tools through visualization control. This way it becomes easier for them to understand the concepts [3]. But when they switch to code writing the interaction ceases and they have to write codes without direct interact with visualizations. They only get to see the visualization after they have finished completely. This way they fail to find out where they have gone wrong and in turn they fail to progress.

In view of Naps et al the overview of best practices that can be used to help in addressing the pedagogical problem of programming can be drawn from increasing the interaction to visualization [6]. Interaction can be increased by including performance information and execution history. This way the students can relate the visualization with the execution history and gain a deeper understanding of the codes that have resulted to a particular visualization change. This doesn't happen in visualization tools as the students use separate text editor away from visualization when writing codes.

Additionally, According to Doherty most students write codes that is syntactically and semantically erratic [7]. The students tend to use feedback from compile in order to revise their code. The feedback may take a while for some IDE's and even when the feedback is provided it is not structured in a way to help novice programmers. But if interaction level is high then a formative feedback for normal execution, errors or anomalies will be provided

right away. So if we combine both the visualization, code writing and visualization control and allow feedback quickly the users will learn to associate all the contents and practice code writing easier. That way the students can transition slowly from visualization to code writing.

2.2. Visualization Tools

In order to develop a system that can narrow down the jump between the using Visualization to code writing it is better to look at the visualization tools and see how they have tried to incorporate writing java codes.

2.2.1. Karel the Robot

This is one of the earliest works that teaches skills of programming and important programming aspects by visualization.[8] The interface is made up of a Robot World that contains a robot called Karel which is positioned at crossing points of vertical and horizontal lines of the Robot World and can move around barriers and manipulate beepers.

The Robot World is composed of Street (vertical) and avenue (horizontal). The Robot World is illustrated using a flat plane of north, east, south and west directions but bounded only by the west and south walls. Some basic functionalities of Karel the robot include moving forward in the direction he is facing, turning left, picking up a beeper, putting down a beeper, and carrying some beepers in his "bag". Also he can find out the presence of nearby walls, can determine if there are any beepers in his bag or if he can pick up a beeper. The system allowed students to write programs that make Karel perform tasks. It has made a

blue print for many of the visualization tools that have been developed including the series of Robot World simulators.

The simulator has an advantage of allowing students to visualize the programs they have written. But in this context of helping the students to write their own codes it is not very useful as the students write codes separate and view the effects of the code they have written later on.

2.2.2. Blue J

Blue J is a Java IDE that has been developed mainly to introduce students to the OO programming using Java [9]. It allows the visualization of the class structure of the system in UML representation and shows how those classes are related. It allows users to interact directly with the classes by creating objects, setting attributes and calling other methods and see the results of that interaction without writing java statements. Then after it allows students to use Blue J text editor to modify Java codes of the existing classes of the project.

Blue J decreases the gap between the visualization and code writing by allowing students to input a limited set of java codes and modify classes that have already been created through visualization interactions.

On the other hand Blue J still lacks the ability of having multiple view of the simulation occurring at the same time (Chapter 2.1). The text editor opens up as a new window and the changes made are seen later after the user saves the editor and compiles the classes. On top

of that Blue J visualisation are only in UML and don't represent visualization as how objects interact.

2.2.3. GreenFoot

Greenfoot is an interactive object world, that aims at motivating students by providing concrete experience with object concepts through interaction and visualisation, using engaging context scenarios, while conveying important object-oriented programming abstractions in the standard Java programming language. [2] It is an educational tool that assists in understanding fundamental object-oriented concepts, and it is highly motivational through instant graphical feedback.

Greenfoot can be used with many different user scenarios, from different topic areas in order to keep the students entertained for example Ants, Karel the robot and lift simulation. As a result of this Greenfoot allows custom environments to be created to specific target groups of novice programmers in order to serve special interest areas. [3]

Greenfoot serves as a good teaching tool to introduce and develop novice programmers to OO programming because it first allows the interaction and visualization of the objects to the users and then makes it possible for user to edit the classes already present. This hides the complexities to the novice programmers for they focus on only writing java codes to edit classes and extends subclasses without worrying about writing the classes from scratch. The green foot text editors also have abilities to check the syntax of the java codes the user inputs which in a way help the novice programmers.

From another angle we can see that the text editor of GreenFoot is loaded externally and code written has to be compiled before being run. Thus, the IDE brings immediate interaction and feedback only when visualization is used but when the users start to write their own code the interactions and feedback is reduced to minimum.

2.2.4. Alice 2

Alice 2 is the most stable of Alice packages at present. It is 3D graphical and interactive micro world programming environment created and distributed by Carnegie Mellon University which has recently gained attention as a gentle introduction to object-oriented programming. [10]. Students use a syntax-free direct-manipulation editor for instantiating objects of 3D animation with simple event handling mechanism. Alice uses a story telling approach and has a number of scenes to allow variation of exercises to the students where by a student's create a story and follow it through as they learn the OO programming concepts.

Alice 2 has very good graphical presentations of the Object Oriented systems. Students can see and manipulate objects directly in the editor using instructions that correspond to java statements without going through compiling complexities like other Java IDE's text editors. In that way a way it is a good teaching tool to introduce students to Object Oriented Concepts.

However Alice has only drag and drop interface does not provide students with much useful experience with syntax as it is syntax free. Hence the students cannot learn how to input java

codes in the text editors. One thing to note though is that the new version of Alice (Alice 3) which is currently not stable will have the feature to allow students to enter java codes and evaluate the syntax.

2.2.5. Dr Java

Dr Java shares the goal of providing a pedagogic environment that minimizes the intimidation factor experienced by new students [11]. It is a text based interface that is simple, interactive, and with a focus on the language. It is composed of an interactions pane, a "read-eval-printloop" (REPL) that evaluate Java expressions and statements interactively. Users type java statements and having it evaluated immediately, without having to write a full Java program. The editor supports multiple documents but does not organize files into projects.

Dr Java has a cleaner and simpler interface that maintains a focus on the Java language. It hides away the complexities of the writing the whole program and provide assistance to students when necessary.

At the same time, the interface is only text based and so it doesn’t address the problem of understanding object orientation by visualization like Alice and GreenFoot.

2.2.6. Jeliot 3

Jeliot 3 is a java teaching tool that uses visualization to aid novice students to learn procedural and object oriented programming. The key feature of Jeliot is the fully or semi-automatic visualization of the data and control flows [12]**.** The latest version of Jeliot 3 has visualization better suited for novice programmers in that it allows a dynamic visualization of objects.

This teaching tool allows the users to interact with the objects through the visualization editors and once the users are confident enough they can start writing their own programs codes in Mini language a called „Not Quite C" (NQC).

The good thing about this teaching tool is that it is easy to use and allows students to learn by doing. Together with that its visual display of the program can be used to facilitate communications about the errors.

Unfortunately Jeliot 3 uses Mini-languages when students enter their own codes to manipulate the objects. This is a drawback because later the students will have to learn afresh the java syntax

2.2.7. Previous Work on RobotWorld Simulator

Since 1999 a number of versions of the Robot World have been built over the years to be used for introducing novice programmers to concepts of programming. Each version has been built in light of the previous versions to improve the capability of the Robot World. .

Since the new system will adopt the functionalities of those systems it is then a good idea to identify their key additional features to the Robot World system.

2.2.7.1 Cannon (1999)

This is the earliest version of the Robot World. His work created in Java AWT followed much on the program of Karel the Robot. It presented well the pedagogical problem of Object Orientation to novice programming and showed how Robot World would help in teaching these concepts [5].

It had the simplest and the most basic functionalities like that of Karel (Chapter 2.2.1). For the program which was created a decade ago, it seemed right for its time. But presently a number of setbacks it has including the limitation of movements of robot to move only forward and only turn right makes its graphical user interface outdated compared to other later versions. And as well it had only an interface where students could input parameters but not construct java statements to control the objects of the Robot World simulator.

2.2.7.2 Chaundry (2000).

Chaundry extended Cannock (1999) work by bridging it to the Web browser [13]. His work also improved the tutorials governing the usage of the Robot World Simulator. The graphical user interface was still the same as that of Cannock. The project emphasis was given to Students learning of Java concepts through tutorials of the Robot World. Since the simulator feature was still the same as that of Cannock then it had the same problem of lacking an interface where students could input java codes.

2.2.7.3 Sally Webber (2001).

This was a much neater work in terms of pedagogical view of programming language as it introduced games feature where the user had to provide some form of coded solution in order to achieve the game's objective [14]. There was HTML tutorial base that student could use to learn java control structure and apply it in the Program interface in the Blue J in order to control the robots. This showed some signs of users starting to write their own codes in the Robot World of which is now the main purpose of this current project.

However the graphical user interface did not change compared to the past projects. It still contained less features and couldn't handle the exceptions well. And still at this stage there was no interface where students could input complete java statements.

2.2.7.4 Adam Fisher (2003)

This work revolved around the visualization of the Graphical user interface and improving the state of information of the components of the simulation [15]. Feedback given to users was increased in that users could easily differentiate when a robot is carrying a cone and when it is not. It also provided diagnostic execution capabilities that gave some feedback to programmers when steps to be retraced in the program are needed.

It brought one strong feature of helping the students practice the concepts of OO. This feature is the algorithm for a robot to find the best path from one location to another. Nevertheless the system was still in adequate because most of these practice were done

interactively using controls an input parameters and in turn the students didn't practice much writing true java statements.

2.2.7.5 Undergraduate (2004/2005) Project

This project introduced a new GUI in Java 2D [16]. The project made a substantial visualization change to the interface and created a new tutorial which combined HTML pages with user-editable classes within Blue J. It also provides Test facilitation of tutorial exercises and feedback to the task performed by the users as in whether they have passed the exercise or not.

This was the first major change to the Robot World as it introduced a completely new look of Robot World simulator. Generally the program was good for introducing students to OO by visualization but like many of it predecessors it didn't include features to allow students to enter 'complete' java statements.

2.2.7.6 Wang (2005)

The project aimed at improving the efficiency of teaching Java [17]. It allowed teachers creating their own set of tutorials for student and allowed students to write java codes in the tutorials to control the robot. The GUI looked like that of the Undergraduate (2004) project with a few added functionalities.

Just like its predecessor the program also was highly dwelt on the visualization techniques as it didn't include interface for students to write their own codes.

2.2.7.7 Penna (2007)

This work introduced a code builder where students could practice the control structures of programming using a form that allowed them to enter parameters to control he simulation [18]. This is a step towards code writing except it doesn't fulfil entirely the concepts of code writer as it doesn't allow users to write java codes. This work also tried to remake the user interface to be more attractive and provided more functionality like the status bar which showed the sell selected at any point in time and deactivated the navigation panel whenever a robot is not selected.

The program made the visualization concepts to be easily understood by the students. The code builder helps the students to understand in depth parameters of methods and variables and explained well the structures of conditional and loop blocks. However the program limited the users to enter only parameters to control the simulation and the students couldn't writer complete java statements.

2.2.7.8 Undergraduate (2007/2008) Project

This work brought about the 3D visualization to the Robot World. It had clear presentation of the objects on the Robot World [19]. Its guide to using the Robot World was well written and easy to use. The tutorial covered a lot of things and took users step by step.

This work had a major advantage in introducing the 3D graphics to make the diagrams attractive and make students pay close attention to the learning as it is interesting to use 3D.

But at the same time, it didn't do much in terms of allowing user to write their own java codes as it only allows them to control the simulation by using the visualization control.

2.2.7.9 Undergraduate (2008/2009) Project

This is the latest work to be done on the Robot World. It has a 3D visualization and a special interface called Code writer where students can write codes to control the object on the simulation [20]. The students can choose to control the simulation by using either the control buttons or the Code writer. It can also run using Both Blue J and Net Beans.

This is the breakthrough for students to write their own codes. However it is still in adequate because the students make a jump from using visualization control buttons straight to writing java codes just like in Blue J. This transition could still be narrowed down to make sure that the students get a chance to use both the visualization controls and the code writer at the same time.

2.3. Summary

Most of the above Visualization tools have included interfaces to allow user to enter text inputs to control the simulation. They have different levels of entering text to manage the visualization. This has ranged from no text interface at all as in Alice, to simple interface where students input only parameters as in Penna's Robot World Simulator with, and finally more complex editors like **GreenFoot** where students can modify completely even the classes that define the visualizations [3, 10, and 18].

The interface like that of Penna's Robot World does not expose the students to much of the code writing. They do not give students room to write complete java syntax codes but rather guide them not to make mistakes. Entering parameters is a step towards code writing as it increases interaction to students but still it is not enough to make the students move from visualization to code writing.

At the same time Interfaces like Blue J and GreenFoot must have a new text editor opened to let users to start inputting java codes. This is then not interactive straight away as the users have to edit the java codes and save them before they see the effects of the codes they have written.

Dr Java visualizes the code while Blue J visualizes the static classes, but to some extent they all improve interaction to students learning programming languages. According to Olam et al, the great strength of Dr Java and Blue J which helps to bridge the gap of jump of visualization to java coding allows beginners to write simple statements and get immediate feedback. [11].

Of the previous Robot World Simulator systems developed so far, The Robot World Simulator of Undergraduate of 2008/2009 has made it possible to input real java codes and observe the visualization effects there. But although it allows java codes to interact with the visualization directly it still has a steep learning curve, because the users can not get the interaction of both the visualization control and code writer at the same time.

The same can be said about DR Java. It allow the user to input basic Java expressions such as 1+1 while at the same time seeing the visualization effects happening which increase the interaction .[12]. The only difference is the type of visualization observed. Dr Java has an advantage of having only simple java codes compared to the code writer. This is far better than Jeliot 3, which visualizes text input from users that are not complete java statements or expression at all as they use mini languages [13].

To transition the novice programmers from visualization to code writing involves gradual development. For most of the visualization tools above, if there is a transition to code writing, the jump is usually high. What is needed from this new Robot World system is to have the transition narrowed down so that the user can gradually move step by step from visualization to code writing. They have to be writing complete java statements and visualize them as in The Robot World of Undergraduate of 2008/2009, or blue J and GreenFoot but the codes should be simple as in Jeliot 3 and Dr Java.

3. System Development.

3.1. User requirements

A series of high level user requirements is listed on table to in order to reach the solution suggested in the chapter 1.3.

Reference Number	User Requirement
UR1.	The system should teach a novice programmer the concepts of objects Orientation using Visualization in 3D.
UR2.	The system should allow the user to see java codes , visualization and human language and the information about the visualization concurrently
UR3.	The system should allow users to control the visualization using the visualization Control button and by writing codes.
UR4.	The system allows users to use two different kinds of Code writers to write the codes. One which will be used with the visualization and the other which will be used alone
UR5.	The system should help users not to run into errors and poor programming styles
UR6.	The system should allow users to learn OO concepts using the tutorials.

Table 3.1 Robot World simulator User Requirements

3.2. Non Functional Requirements

Like any other system the efficiency and effectiveness of the Robot World system will rely on key non functional requirements. These are tabulated on the following table

Reference Number	Non Functional Requirement
NFR1.	The system must be able to run on Windows and Linux Operating Systems
NFR2.	The system should be able to run in Java 3D.
NFR3.	The system must never crash or fail throughout the simulation
NFR4.	The system has to be easy to maintain
NFR5.	The system must allow future improvement to be made possible.
NFR6.	The simulator which is easy to use and easy to understand.

Table 3.2 Robot World simulator Non Functional Requirements

3.3. Overall Design Analysis

3.3.1.1 Feature Analysis

In order to satisfy the requirements as stated in table 3.1, the Robot World simulator system to be developed must have at least the following features.

3.3.1.2 Visualization Grid

This is to be the place where users see all the objects and actions that take place. The visualization is in 3D graphics. The grid is composed of 6X6 square matrix cells that can contain objects. The objects are cone, robot and barrier.

3.3.1.3 Visualization Control buttons

These are to be buttons that manage the objects of visualization. The actions which can be performed are to create the three objects, Move the robot around to manipulate the cones and clearing the contents of the visualization grid.

3.3.1.4 Interactive Code Writer

The interactive code Writer handles the Java codes of the Robot World simulator. It allows the input and output of the java codes to control the visualization. It is used side by side with visualization control buttons.

3.3.1.5 Programmed Code Writer

The programmed Code writer exists on its own without the visualization control buttons. This type of code writer allows many more types of statements to be written including conditional statements and while loop.

3.3.1.6 Status Bar

The status bar displays the active cell selected. Together with that it also displays from time to time some necessary information for example when the user loads a tutorial.

3.3.1.7 Text Viewer

The text viewer outputs the pseudo codes. The pseudo codes are similar to English phrases so that they can easily be understood by users in case they still haven't grasped the java syntax. The text viewer displays the pseudo codes whenever the action occurs on the Robot World simulator.

3.3.1.8 Tutorials

The tutorials are web pages create din HTML that teach the users about the OO programming in Java.

Table 3.3.1 summarizes the system features above that will serve the user requirements

Reference Number	**User Requirement**	**System Feature**
F1.	UR2	Overall Robot World Graphical User Interface(GUI)
F2.	UR1,2,3,4	Visualization Grid
F3.	UR2,3	Visualization Control Buttons
F4.	UR2,3,4	Interactive Code Writer
F5.	UR2,4	Programmed Code Writer
F6.	UR2	Text viewer
F7.	UR2	Status Bar
F8.	UR5	Syntax Parser
F9.	UR6	Tutorial

Table 3.3.1 Robot World Simulator features

From chapter 1.3 the basic idea behind is to make sure that the novice programmers gradually migrate from Visualization to Code writing. The first to do is to allow the Robot World to display as many features as possible so that the users can associate perspectives of the simulation.

To allow this the Robot World will have in one window all the necessary features present t make users" associate visualizations, java codes and English words. Therefore when the users first loads the Robot World simulator, it will be better if they see the visualization grid, visualization controls buttons interactive code writer and text viewer all together To allow the move of code writer to be swift it will be made possible to have another version of the same Robot World simulator which will not have the control buttons so that the user can only control the objects by input the codes. This way the user will first start of by using both the control buttons and java codes to control the simulation before he goes on to use only java code writing to control the simulation.

The code writers will have the ability to check for syntax errors and poor programming styles of the code the users enter before they are executed. One thing to bear in mind is when best to give users useful feedback when they run into errors. The feedback on the errors will be given momentarily on the same window and the execution stopped and waits for the user to correct the errors before continuing as suggested in chapter 2.1. Once the user has got the statement in correct java syntax then the code will be executed. The programmed code writer is to able to copy the contents from interactive code writer so as to help the user have a starting point when he wants to use only java codes without control buttons.

In addition to that it will be possible for the contents of both the code writers and the text viewer to be saved for future references.

The Robot World simulator will also contain tutorials to teach students the OO orientation. The users will be able to open the tutorial from the Robot World simulator in a web browser. The users will open the tutorial in a web browser and read through even when they have not opened the Robot World simulator.

With this in mind there will then be a menu item that allows users to move from Robot World simulator with Interactive code writer and another item of Robot World simulator with programmable code writer. In addition to that there will also be a menu item to allow users to open up tutorials for OO programming.

3.3.3. Design Considerations

From chapter 1.3 the Robot World will include the best of the previous Robot World systems as well as incorporate features of a good visualization tool that helps the user to transition gradually from visualization to coding. There are two available approaches to implement the system. One is to develop everything from scratch and the other is to use one of the previous Robot World simulator systems and change some features in order to get the system that works in a preferred way. For each one of them there is a high risk that is associated with choosing it.

If the approach of developing a system from scratch is taken, there will then be a total control of whole system development. There will be new design of the system and code

writing of all the associated features. However it might take a long time to design the whole system and implement it from scratch and there is a high risk that it might not be finished in time.

On the other hand if one of the previous systems is reused then it will be easier because most of the basic functionalities of Robot World simulator system needed, will already have been implemented and all that will be left will be designing and implementing those parts that weren't there in the first place. But again, with this approach, time will have to be spent to understand the source code of the system and there is a risk that I might not fully understand how it works and might get stuck at some points. Since there is a tight time schedule then it is better to reuse the code that has already basic functionalities and change it to fit to the requirements identified in chapter 3.1.

The new Robot World simulator will reuse the Robot World simulator system by the undergraduate of 2008/2009. This has been chosen because first and foremost it is the latest version of the Robot World simulator. And since the Robot World simulators have been built on top of the previous ones it will have contained most if not all the features that were in previous versions as well. As well the 2008/2009 undergraduate project will be a good starting point because it already has a code writer designed and can visualize object in 3D. Thus less work will be done in just implementing the code writer in the way this project wants the code writer to be used.

As many of the basic features that the Robot World simulator is going to posses have already been implemented by the Undergraduate project then it is a good point to find out

how the features of the new Robot World compare to the previous one. Table 3.3.2 compares the features of the new Robot World simulator against those of the system of undergraduate project 2008/2009.

Feature	**Undergraduate Project 2008/2009**	**New Robot World Simulator**	**Reason**
F1	Switch between the Code Writer , control buttons and the location finder	Both displayed at the same time	Make the user see the code writer and Control buttons and choose the one to use
	Contains the location finder as a menu item	Location finder replaced by the Status Bar	No need for a separate location finder menu as the status bar will be there
	Has no Status Bar	Has a Status Bar	Displays constantly the location of the selected cell
	Pop up when anomalies appear during robot movement	Provide pop ups and animations when anomalies appear during robot movement	Addition of animation to keep the user entertained
	The buttons and the code writer control	User able to control the same object using buttons	User can start using the control at one time

	object at different times	and code writer interchangeably	and switch to code writing on the same object
F4	No Interactive Code Writer	Has Interactive Code Writer	Code writer is seen from start when the Simulation is opened up
		Displays Java Codes output when students use Visualization control buttons	To allow the students to make relationship between java codes, human language and visualization
F5	Code Writer has a simple checker	Code Writer has a complex syntax checker	To allow the system to pick up errors and suggest solutions
		Can load contents from the Interactive Code Writer	To give users a starting point
F6	Output Java Statements	Output Pseudo codes	Close to human language and code writer already has Java statements
F7	Output Information	Displays the active cells	Always know where

			the student is in the visualizations
F8	Scans user inputs and alerts the user when he has made an error	Alerts the user where an error is made and provide assistance	Picks up errors and suggests corrections

Table 3.3.2: Comparisons of features of the new Robot World simulator to the old Robot World Simulator

3.3.4. Development Planning

The methodology that will be followed in this project will be the Iterative development model. At first it was proposed to use Waterfall development model as it allows the system to be developed within a short time but after a literature review and preliminary analysis of the system it was decided that Iteration would better suit the development process.

It is appropriate to use Iterative approach in the Robot World simulator because it allows the systems to be developed in a number of steps with each end product representing a working system that can satisfy a subset of the basic requirements.

This will address the question of time limit. The time to develop the system is short and therefore by having to develop the system in a number of iterations it will make it possible to have some few full working features by the deadline.

The user requirements will be extended to detailed functional requirements. The mandatory and significant functional requirements will be implemented first and the minor requirements will be implemented later. There is a chance that some functional requirements won't be implemented due to time limit and some will evolve as development goes on.

The code writer will have to be implemented first as it is the significant focus of the project. This will be followed by the syntax checker and tutorial. Lastly we will end up with the Animations as it is a minor requirement.

Both Java Netbeans and Eclipse provide good IDE's for implementing systems that require good Graphical user interface. Since the system will reuse the previous codes then it will be convenient to implement it using the Netbeans as it is the Java IDE that was previously used to develop the existing system.

The system development will therefore have two Iterations. The first Iteration will consist of creating the Graphical User Interface for the system and the Second Iteration will be dealing with Syntax parser Tutorial and animation of the Robot World simulator.

3.4. First Iteration

3.4.1. Aim

The first Iteration focuses on embedding the code writers to the Graphical user interface (GUI) of the Robot World Simulator system and make sure that it can allow user to control

the visualization by both visualization control buttons and the code writers. It is assumed that the codes entered by the users at this point contain no errors.

3.4.2. Functional Requirements

Table 3.4.2 lists the functional requirements to make sure the code writer is embedded and works without problems

Reference Number	**User Requirement**	**Functional Requirement**
FR1	UR1	Allow users to see all the features of the Robot World (3dVisualization Panel, Code writer, control buttons, status bar and text viewer) at the same window all the time
FR2	UR3	Allow the user to navigate through code writer and Control buttons to control the input of the Robot World
FR3	UR2	The user can visualize on the Robot World the effects of the input actions put in the control button and code writer
FR4	UR3	Allow the users to manage the robot objects in Robot World by clicking on the Control buttons
FR4.1	UR3	Create robot, cones and barriers
FR4.2	UR3	Move the robot forward if there is no barrier in front
FR4.3	UR3	Move the robot backward if there is no barrier behind

FR4.4	UR3	Turn the robot left
FR4.5	UR3	Turn the robot right
FR4.6	UR3	Pick up a cone
FR4.7	UR3	Drop a cone
FR4.8	UR3	Fail to move the robot in the cell where there is a barrier
FR4.9	UR3	Delete robot, cone and barrier from the Robot World
FR5	UR3	Allow the user to manage the robot objects in Robot World by writing codes in the code writer
FR5.1	UR3	Create robot, cones and barriers
FR5.2	UR3	Move the robot forward if there is no barrier in front
FR5.3	UR3	Move the robot backward if there is no barrier behind
FR5.4	UR3	Turn the robot left
FR5.5	UR3	Turn the robot right
FR5.6	UR3	Pick up a cone
FR5.7	UR3	Drop a cone
FR5.8	UR3	Fail to move the robot in the cell where there is a barrier
FR5.9	UR3	Delete robot, cone and barrier from the Robot World
FR6	UR3	Allow the user to control objects of the Simulation by using control buttons and code writer simultaneously
FR7	UR2	Allow the user to see the input actions from control buttons in the Code Writer as java codes
FR8	UR2	Allow the user to see pseudo codes of the input of code

		writer in the Text viewer
FR9	UR2	Allow the user to constantly see the location of the selected cell in the status bar
FR10	UR4	Allow the user to switch between the Code writers using the Menu bar

Table 3.4.2 Iteration 1 Functional Requirements

3.4.3. Detailed Design Analysis

The GUI will be divided in two halves, input and output parts. The left half will contain the input and right half will contain the output. For the input half, since the students begin to use the system by the visualization control button then this will be displayed at the top most left half and the lower half will contain the code writer.

On output part the visualization grid will be displayed on top most. Below the Robot World there will be controls for speed of the execution followed by the status bar which will display the active cell. The last component of the output half will be the code window that will display. The visualization grid will be larger than other output code window because it has to visually display what the user has typed. Here is the diagram that shows how the GUI interface of the system will look like.

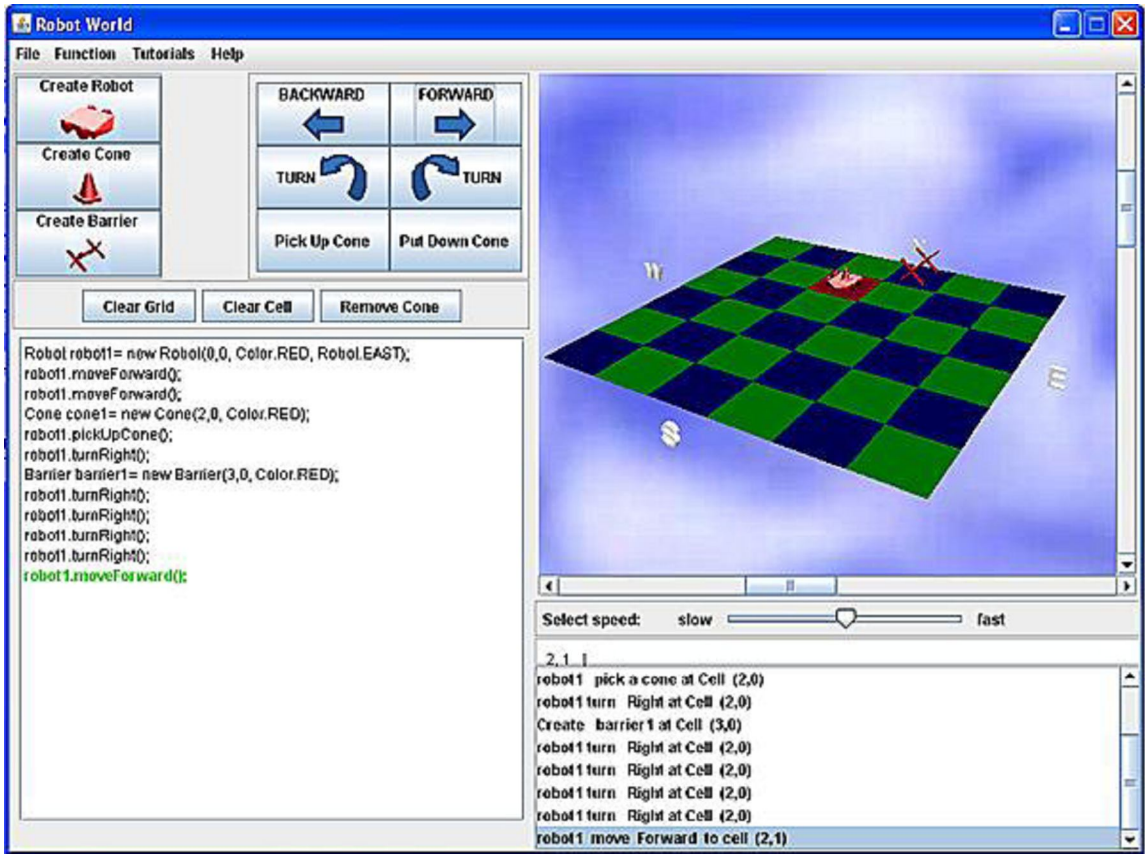


Fig 3.4.3.1 Robot World Simulator with Interactive code writer.

The system will have an option of choosing an interface where the students will enter codes alone to control the robot and therefore the control buttons won't be needed. In this part of the code writer the users can write as many lines of code to manage the objects as they want. Since there will be only one input source it will be okay for the code writer to fill the whole input area. The output part won't change and will still consist of the Visualization Panel, Status bar and the Text viewer. Here is the sketch of the Interface.

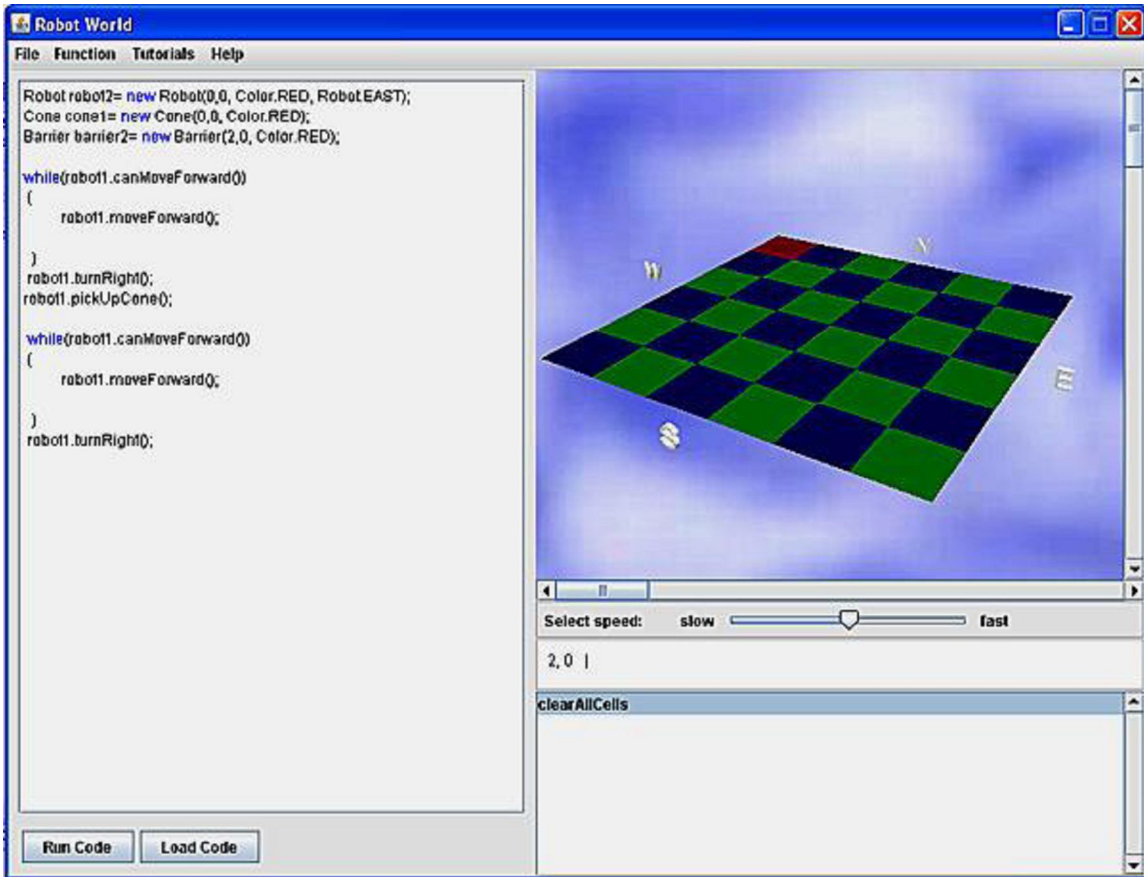


Fig 3.4.3.2 Robot World Simulator with Interactive code writer.

The text viewer will output pseudo codes similar to English language in order for users to understand what is going on. Each time a user does something on the simulation messages will be displayed here on the viewer one line at a time to show which line event has occurred.

The status bar will keep track of the active cell. This will be embedded below the Robot World visualization panel and updated each time an event occurs in the Robot World.

Appendix C shows the class diagram of how these features will be embedded to produce the desired GUI.

3.4.4. Implementation

Java Swing in Netbeans will be used to implement the desired GUI. Since the development of the new Robot World simulator will be based on the previous version some of the features will be remain the same. Looking back at the feature summary table 3.3.2, no changes will be implemented to Visualization 3D and the Visualization Control buttons. However the Interactive Code Writer, Programmed Code writer Text viewer and the Status Bar will change to suit the new Robot World simulator.

Messages to be output by the status bar and the text viewer were passed as a string and the active cell to be displayed was passed as coordinate argument.
The list panel was used to output text in text viewer that has the capability of making the most recent statement has to be highlighted and become different compared to others.

Among the major concern was to how to implement the interactive and programmed at the using only one type of code writer. One object was created and given key listeners and sizes that were different. The interactive code writer has a small size and was given an Enter key listener and the programmed code writer was given large size without key listener. These would change every time the user switches to different type of code writer.

The implementation could also have been achieved by having two separate classes of code writers with different capacity. But then this would mean code duplication. So at the end the idea of having one code writer was implemented. This represented maturity of code writing as it requires the close follow up on the classes whenever it was used.

3.4.5. Testing

For the complete testing both the white box and Black box testing have been carried out to make sure that the system is working correctly. If errors were found they were resolved and the process of testing was restarted from scratch to ensure the changes successfully solved the problem.

3.4.5.1 Units testing

The unit testing has been carried to make sure that those parts that were added to the original classes are okay and functioning correctly. This used the white box principles and J Unit has been used for testing the parts which were added or changed in order to satisfy the functionalities of iteration 1. The tests were all successful for these parts.

3.4.5.2 Functional testing

Functional testing which is concerned with the “correct” functionality of the package was undertaken using a black box approach. The system was later tested to see if all the functionalities can be fulfilled. A series of test cases case was used to try to establish each requirement. Refer to the appendix E for the full list of the testing table and results.

3.4.6. Revision Analysis

The first Iteration has been successfully implemented. The user can use all features and be able to see multiple views of the execution. However though some bugs that would interfere with the further development and functionality of the system were resolved successfully but there are still some for bugs which required lengthy analysis. These will not interfere with

system development or key functionalities of the system so it was better to leave them to be solved in the future.

Reference Number	**Bug Description**
B1	Two objects can share the same name
B2	The code writer stops displaying text after the user has used it more than 6 times
B3	The colouring of the active cell the Robot World is incorrect when there are 2 or more robots present

Table 3.4.6 List of Iteration 1 Bugs

3.5. Second Iteration

3.5.1. Aim

The second Iteration will focus on mainly two things. The first one will be to make the Code writer more intelligent and give more feedback to the users. And the second will be to focus on making the tutorial for students in order to use it well to understand the concepts of object oriented programming languages.

3.5.2. Functional Requirements

Functional Requirement	**Test**	**Expected**	**Actual**
FR11	The system detects where the error is in the piece of code entered by the user	Pass	Pass
FR11.1	The system detects Creation Statement Errors	Pass	Pass
FR11.2	The system detects object Method Errors	Pass	Pass
FR11.3	The system detects Conditional Structure Errors	Pass	Pass
FR11.4	The system detects Loop Structure Errors	Pass	Pass
FR11.5	Feedback given to users to show them where the errors are and how to rectify the error	Pass	Pass
FR11.6	Color the string of error code with Red Error	Pass	Pass
FR12.1	Introduction page describing the control structures and other object source of information of OO programming	Pass	Pass
FR12.2	User provided with Sequence Control Structure Tutorial	Pass	Pass
FR12.3	Users provided with Conditional Control Structure Tutorial	Pass	Pass
FR12.4	Users provided with Loop Control Structure	Pass	Pass
UR13	Sound to make sure the user knows the code has failed	Pass	Pass

Table 3.5.2 Iteration 2 Functional Requirements.

3.5.3. Detailed Design Analysis

The syntax checker will keep track of the user codes input and provide assistance when the user makes a syntax error. In this way the user will always know where he has made a mistake. The line with the error problem will be highlighted in red color and a suggestion feedback of how to correct the error will be given to the user as a pop up. The code written will not run until all the errors have been rectified.

Most of the errors will be captured by the code writer but it is not possible to be able to find all the exact errors the users have made. The lists of errors that can be trapped by the syntax checker are listed in the Appendix G.

The tutorial on how to use the Robot World Simulator will be written in html so as it can be used in web browsers. The tutorials will be loaded from the menu bar. The tutorials open up as a side frame and the users can read through while continuing using the simulation. Some tutorials have a starting point and once opened they will load object on the Robot World so that the users can continue manipulating them. When the tutorials are opened a tutorial menu will appear to track the progress of the users and see if they have completed the tutorials successfully.

The simulator has to play a sound whenever there is an anomaly to the normal execution of the Robot World simulator. The anomalies from which the sound will be produced are listed in the appendix H.

Appendix D lists the detailed class diagram of how the design of the syntax parser and the tutorial was achieved.

3.5.4. Implementation

The implementation of the syntax checker has been by far the toughest part of the project. The idea behind was to capture as many errors as possible and yet to use efficient and tidy code.

The first approach that was desirable was to go through word after word and try to check if it is correct according to writing the structure of a particular statement. If the word has syntax error or poor programming styles then we simply output the possible errors and feedback. This would make sure every phrase of the entered code is checked. However it is only applicable if the input to look for errors is small. If the source of statements to check for errors is large then it might require a volume of code to be written.

The second approach that could also be taken was to use regular expression. [21,22] Regular expressions were created to describe different type of errors and programming styles that the user could have run into and stored in HashMap. For each regular expression stored as a key there is a corresponding feedback message set as a value. The statement with error was broken into a number of word tokens and stored in an array. Then the tokens have to then be matched against the regular expression. When they match the feedback will be output using that regular expression. At the end if there is no match then the default feedback is given to the students. This was advantageous as it keeps the code tidy and allows the capture of more

errors with few java codes. But at the same time it is hard to implement and requires a lot to be done and there is chance that some words might be overlooked.

At the end the syntax parser was implemented using the regular expression due to the fact that the code was tidy and would allow easy expansion of error checking mechanism in the future.

For the tutorial part the hard thing was to link the Robot World to the tutorials the students has to go through. The tutorials were created as web browsers using HTML. These were embedded to the Frame of Java swing and in that way they could be called from the Robot World using a Menu button.

The trick to add the sound to the system whenever an error occurred was where in the Robot World class to add the method to play the sound. One way was to add the sound clip separately for every action that has been performed on the Robot World and put a condition that if the action usefully completes then the system shouldn't play any sound. The other option was to make sure that the sound clip is added at the class that controls the visualization change. In this way once the visualization changes which do not complete the sound clip should play to signify the errors.

The later method was implemented because it first allowed only one method to be added to the entire package and hence reduced code duplication and at the same time it was able to capture all the irregularities of the Robot World and play the sound clip.

3.5.5. Testing

Testing was redone for the second iteration after it was implemented. It used the same approach of white box and black box to be able to test both the quality of codes and the fulfilment of the functional requirements. New test cases were created to fit only to the focus of second iteration requirements

3.5.5.1 Unit testing

White box testing was done by Unit testing. The unit testing looked at those parts of the system that were added only in Iteration 2. This was done using White box system where the methods that were added in the second Iteration including the check Syntax were evaluated to find if they contain any errors. At the end, the systems were found to contain no errors as the source code functioned correctly.

3.5.5.2 Functional Testing

This used the black box testing .As in first iteration after the system was developed it was tested against the functional requirements to see if it meets all the needs. Most of the user requirements were satisfied but they were a number of anomalies in checking for errors in the condition and loop structures as tabulated in the Appendix E.

3.5.6. Revision Analysis

The system has been implemented to the original plan but still just like in the first Iteration at the end of this part as well there were some bugs that the system contained. Table 3.5.6 shows the bug which were not resolved.

Reference Number	**Bug**
B4	The syntax checker doesn't trap all the errors
B5	The errors location can sometime mislead
B6	The tutorial image when opened from Java Robot World simulator GUI are distorted

Table 3.5.6: List of Iteration 2 Bugs

4. Evaluation

The Robot World simulator is designed to be an effective teaching tool for novice programmers. In order to find out if it has been successful it was vital to revisit the user requirements and find out if it can serve the user requirements it was originally developed for. To achieve true evaluation of the user requirements, the users had to be approached to evaluate the system and see if their requirements were fulfilled

4.1. The Evaluation Strategy

The tutorial is lengthy and requires a person to settle down for quite some time in order to complete all the tutorials and get a deep understanding of the Object Orientation principles. It was then not possible at this evaluation stage to ask volunteers to do the whole tutorial in order to evaluate it. To overcome that deficit a set of tutorial in terms of tasks were created and users asked to perform those tasks to completion and give feedback on how they found the Robot World simulator to be. This can be obtained by running the evaluation copy. After attempting a set of 3 tasks the evaluators were provided with open mind questions in order express their views on how they see the program and not how it is used.

Quality and not quantity is the key attribute for better evaluation and therefore the evaluators were chosen from three groups of novice programmers, experienced programmers in order to get a good feedback it is better to include the opinions of different parts. The evaluators were left to go through the tutorial on their own and were observed closely and given assistance when and where necessary.

4.2. Feedback from Evaluators

The evaluators looked at different perceptive of the system and raised a number of issues. For those issues that were important for this particular project and required immediate actions they have been changed and for the other issues that were not the primary concern of this project and require in depth analysis have been left for future improvement.

4.2.1. Using Code Writer

The evaluators felt that as the code writer was output codes it was to keep track on which code is the most recent one and needed to be associated with the visualization change that has occurred. A suggestion was then given to make the most recent code be different compared to other codes that were there before. This was seen as vital change and was implemented right away.

In addition to that, it was suggested to have auto complete feature to finish up java codes for novice programmers. However this will hide the possibility of students running into errors. Therefore it was seen that the feature that may require in depth analysis. It wasn't implemented and left for future work that will be done on the Robot World simulator.

4.2.2. Text Viewer

The text viewer has been seen to output human language. The interface was criticized as always displaying words and it might be hard to follow through and discover which line was showing the visualization change that has occurred. Because this change was vital to the requirements of this project then the change was implemented right away.

4.2.3. 3D Visualization

The visualization panel is the part of the Robot World that received much criticism and applause. The experienced programmers liked the way it was implemented and were happy to use it but the novice programmers raised a number if concerns.

First and foremost the Directions are not clear. The students felt that the directions given as North, South, East and West Direction should reflect the normal directions. This way it will be much easier for the students to control the objects according to the directions.

Of the other things the students commented on was the Robot World origin. The origin (0, 0) was on the right and the scrollbars were located to start at top most right for the y axis and left bottom corner. This confused students who are familiar to using graphs in Cartesian coordinate.

The last suggestion given for the visualization panel was on how to increase the interaction of users by allowing the cells to be clickable. Evaluators felt that if the cell were editable and allow actions then the user could interact more freely with the visualization and be able to learn better.

4.2.4. Language

The language used in various parts of the system was seen as the problem. In the task to be carried it was still unknown what the term „Robot World" represents. In some cases it

represented the area of visualization of the objects while in some cases it represented the whole simulator.

On top of that the language used in the tutorial was also seen as to contain a lot of technical terms that may not be appropriate for the novice programmers. Instruction like “create a robot in cell” 0, 0 was seen as if is too technical. This hadn’t had such a big impact and was left for future analysis and implementation.

Appendix I summarizes the suggestion that have been obtained from the evaluators and the state to whether they have been implemented or not.

4.3. Future Improvements for RobotWorld Simulator

The Robot World simulator has so far been able to teach the concepts of creating objects, methods and the control structures of conditional and loop structures. However there are some aspects of object orientation concepts and user requirements that have not been achieved by this system. These concepts are Encapsulations, Arrays and Inheritance. It is good to introduce these concepts so that the students can understand them early as they will be useful in the future of their programming.

The 3D graphics that are currently displayed on the simulator are good but not efficient. The directions are still confusing and sometimes it is hard to know where north and south really are. Much work still needs to be done on the visualization to make it more attractive and efficient.

In addition to this, the syntax checker the Robot World simulator cannot find all the errors that can be made by novice programmers. It is really hard to find all the errors that can be made by the students. Some even write things that are not known even in any language and it is hard to trap all these errors. Thus the future work should be done on this area to expand the scope or errors to be trapped by the code writer.

Most of the work done in Robot world has been directed to the interface and the tutorial which should be taken as the main part has been ignored. Therefore future work to be done on Robot World simulator should be directed to the tutorials in order to address the pedagogical problems of teaching java to novice programmers.

4.4. Evaluation Summary

The approach of choosing wide variety of evaluators has resulted in useful feedback that will be incorporated in the Robot World teaching tool in order to make it more user friendly and effective.

From the test cases most of the functionalities of the Robot World system as projects were validated. What was left was trying to see if those functionalities would serve the user requirements of teaching OO concepts to real users. This was proved by the evaluators who felt the Robot World system to be very useful. The evaluators liked the fact that the teaching tool had 3D graphics and that it was easy to use. In a large part the Robot World simulator was found to teach well the principles of OO in java and transition students from Visualization to code writing if well followed.

Although the simulator has been found to contain many features that can help introduce the students into the concepts of OO, it can still be see that the simulator is limited in a number of ways when it is being used as observed by the evaluators. They pointed these out and the suggestions which were felt to be important have been added to the suggestions for future work on Robot World Simulator for Java.

5. Conclusion

When the project first started its main aim was to create an instance to help the students input java codes. But after a review of the previous work done on the Robot World and on the visualization teaching tools, the aim was extended to include a gradual transition to code writing with an additional feature of assisting the novice programmers once they run into errors or poor programming styles.

The original aim of the project has thus been achieved as a new Robot World system that has a gradual transition from using control buttons to complete code input has been created. A series of tutorials have been prepared which can direct the students to learn the concepts of OO programming on their own.

The source code has been extended neatly from the precious source codes and allows room for improvement in case there is a need to follow up on this work.

The two types of code writers allow the users to gradually learn how to write proper java codes step by step. The feedback given to students is resourceful enough to direct the users to become better programmers.

In conclusion a new and efficient Robot World simulator has been achieved which can be seen to narrow the transition gap from OO introduction through visualization to code writing and with which there is no doubt that it will be beneficial to teaching OO programming to novice programmers.

6. Bibliography

[1] Kaasboll, J, *Learning Programming*, University of Oslo 2002.

[2] Kölling, M, Henriksen, P,Game programming in introductory courses with direct state manipulation, *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, June 27-29, 2005.
http://www.greenfoot.org/papers/2005-06-ITICSE-greenfoot.pdf

[3] Kölling, M, Henriksen, P, Greenfoot: combining object visualizations with interaction, *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, October 24-28, 2004, Vancouver, BC, CANADA .
http://www.greenfoot.org/papers/2004-10-OOPSLA-greenfoot.pdf

[4] Chen , Z , Marx, D, Experiences with Eclipse IDE in programming courses, Journal of Computing Sciences in Colleges, v.21 n.2, p.104-112, December 2005
http://portal.acm.org.chain.kent.ac.uk/citation.cfm?id=1089068&dl=GUIDE&coll=Portal&CFID=51627811&CFTOKEN=33970768

[5] Cannock R. "Robot World Simulator for Java", *MSc dissertation*, University of Kent, 1999.

[6] Naps, T, Rößling, G, Almstrum, V , Dann,W. , Fleischer, R , Hundhausen ,C , Korhonen, A , Malmi, L , McNally, M , Rodger, S , Velázquez-Iturbide ,J. A. Exploring the role of visualization and engagement in computer science education, Report of the Working Group on "Improving Educational Impact of Algorithm Visualization . 2002
http://portal.acm.org.chain.kent.ac.uk/citation.cfm?id=782998&dl=ACM&coll=portal&CFID=50041492&CFTOKEN=20185429

[7] Doherty, L, Lougheed,P, Brokenshire,D, Jordanov
Jordanov, M, Rao, S., Shakya, J, Recognizing opportunities for mixed-initiative interactions based on the principles of Self-Regulated Learning, *AAAI Fall Symposia on Mixed-Initiative Problem-Solving Assistants*, Crystal City, VA, USA, accepted for publication.
http://www.stanford.edu/~smenon/professional_files/publications/recognizing_opportunities_for_mi3.pdf

[8] Becker, B. W, Teaching CS1 with karel the robot in Java, *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*,
p.50-54, February 2001, Charlotte, North Carolina, United States.
http://www.cs.uwaterloo.ca/~bwbecker/papers/sigcse2001/karel/

[9] Kölling M, Rosenberg J, An object-oriented program development environment for the first programming course, *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, p.83-87, February 15-17, 1996, Philadelphia, Pennsylvania, United States

http://portal.acm.org/citation.cfm?id=236514&dl=GUIDE&coll=GUIDE&CFID=51122965&CFTOKEN=41377118

[10] Mullins ,P, Whitfield ,D. Conlon ,M, Using Alice 2.0 as a first language *Journal of Computing Sciences in Colleges*, Volume 24 , Issue 3 (January 2009) Pages 136-143 **http://delivery.acm.org/10.1145/1410000/1409900/p136mullins.pdf?key1=1409900&key2=4537852521&coll=GUIDE&dl=GUIDE&CFID=52228163&CFTOKEN=11730358**

[11] Olan ,M, Dr. J vs. the bird: Java IDE's one-on-one, Journal of Computing Sciences in Colleges, v.19 n.5, p.44-52, May 2004 **http://portal.acm.org.chain.kent.ac.uk/citation.cfm?id=1060089&dl=ACM&coll=portal&CFID=50041492&CFTOKEN=20185429**

[12] Moreno, A , Myller ,N, Sutinen, E , Ben-Ari, M, Visualizing programs with Jeliot 3, Proceedings of the working conference on Advanced visual interfaces, May 25-28, 2004, Gallipoli, Italy**. http://delivery.acm.org/10.1145/990000/989928/p373-moreno.pdf?key1=989928&key2=5568852521&coll=GUIDE&dl=GUIDE&CFID=52228163&CFTOKEN=11730358**

[13] Chaundry, M.K, "Robot World Simulator in Java", *MSc dissertation,* University of Kent, 2000.

[14] Webber, S, "Robot World Simulator in Java", *MSc dissertation*, University of Kent, 2001.

[15] Fisher, A. J, "Robot World Simulator in Java", *MSc dissertation*, University of Kent, 2003.

[16] Turner,G, .Watling, R, Lee, H. G, "Robot World Simulator in Java", *Undergraduate Project*, University of Kent, 2004.

[17] Wang, Z, "Robot World Simulator in Java", *MSc dissertation*, University of Kent, 2005.

[18] Penna,R. "Robot World Simulator in Java", *MSc dissertation*, University of Kent, 2007.

[19] Addison J, Ademol,J, Shermohammad, A, Joe Pang, H.W, Pang , K W H, "Robot World Simulator in Java", *Undergraduate Project*, University of Kent, 2008.

[20] Ihedioha, N, Lin, W Liu, J, Phipps, K, "Robot World Simulator in Java", *Undergraduate Project*, University of Kent, 2009.

[21] Kelleher ,C, Pausch ,R, Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers, *ACM Computing Surveys (CSUR)*, v.37 n.2, p.83-137, June 2005.
http://www.cs.cmu.edu/~caitlin/papers/NoviceProgSurvey.pdf

[22] Deitel, P. Deitel, H,*Java How to Program, 7th Edition*, Pearson Education Inc, 2007.

[23] Turner, J. A, Zachary ,J L., Javiva: a tool for visualizing and validating student-written Java programs, *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, p.45-49, February 2001, Charlotte, North Carolina, United States.
http://delivery.acm.org/10.1145/370000/364535/p45 turner.pdf?key1=364535&key2=6681678321&coll=GUIDE&dl=GUIDE&CFID=29532977&CFTOKEN=16906864

[24] Reis, C., Cartwright, R, A friendly face for Eclipse, *Proceedings of the 2003 OOPSLA workshop on eclipse technology exchange*, p.25-29, October 27-27, 2003, Anaheim, California. **http://delivery.acm.org/10.1145/970000/965666/p25-reis.pdf?key1=965666&key2=7132678321&coll=GUIDE&dl=GUIDE&CFID=28776162&CFTOKEN=27156998**

[25] n, also

[26] Niemeyer, P, Jonathan K, *Learning Java 3r Edition*, O"Reilly, 2005.

[27] Fenner, T, Loizou, G, Mannock, K, Vee, M.C (2009). A Simple Interactive Development Environment for C# available at **http://www.dcs.bbk.ac.uk/~keith/research/sIDE/paper/jicc7.pdf last accessed on 03rd April 2009**

7. Appendices

Project	**Strength**	**Weakness**
Cannon	• Explain clearly the objectives of the project. • textual representation of the robot world	• Primitive Code • Outdated • Contains less features • Doesn't contain Code Writer
Sally Webber	• Tutorial based learning(The first sign of code writer) • Warning messages	• Almost the same as that of Cannon • Still contains less features • Doesn't handle exceptions well (Program failure) • Doesn't contain Code Writer
Adam Fisher	• state information of the robot • algorithms to which it can find the best path from one location to another	• • clicking on a cell has no effect • Doesn't contain Code Writer
Undergraduate Project 2004/2005	• Good User Interface • Well commented.	• Less feedback to users • Doesn't contain Code Writer
Wang	• Good feedback to users when error is encountered	• Doesn't contain Code Writer
Penna	• Easily understood • Contains the code-Builder • Well documented source code	• Doesn't contain Code Writer • unnecessary feature- Code Builder

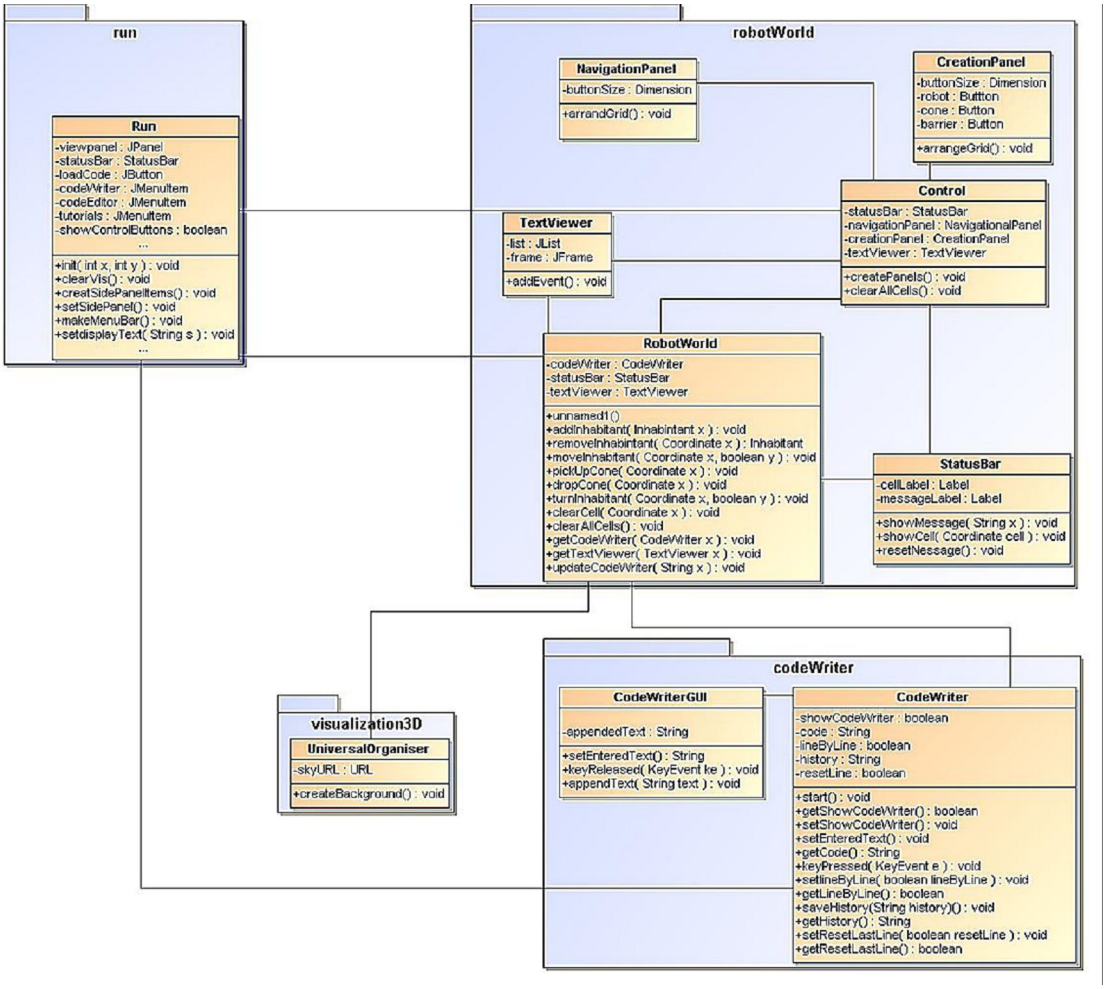
		• Does not have the tutorial
Undergraduate 2007/2008	• Used 3D Visualizations.	• Does not contain code writer
Undergraduate 2008/2009	• Used 3D Visualizations. • Contains the code writer	• Contains the tutorial

Appendix A: Summary of Previous Work on Robot World

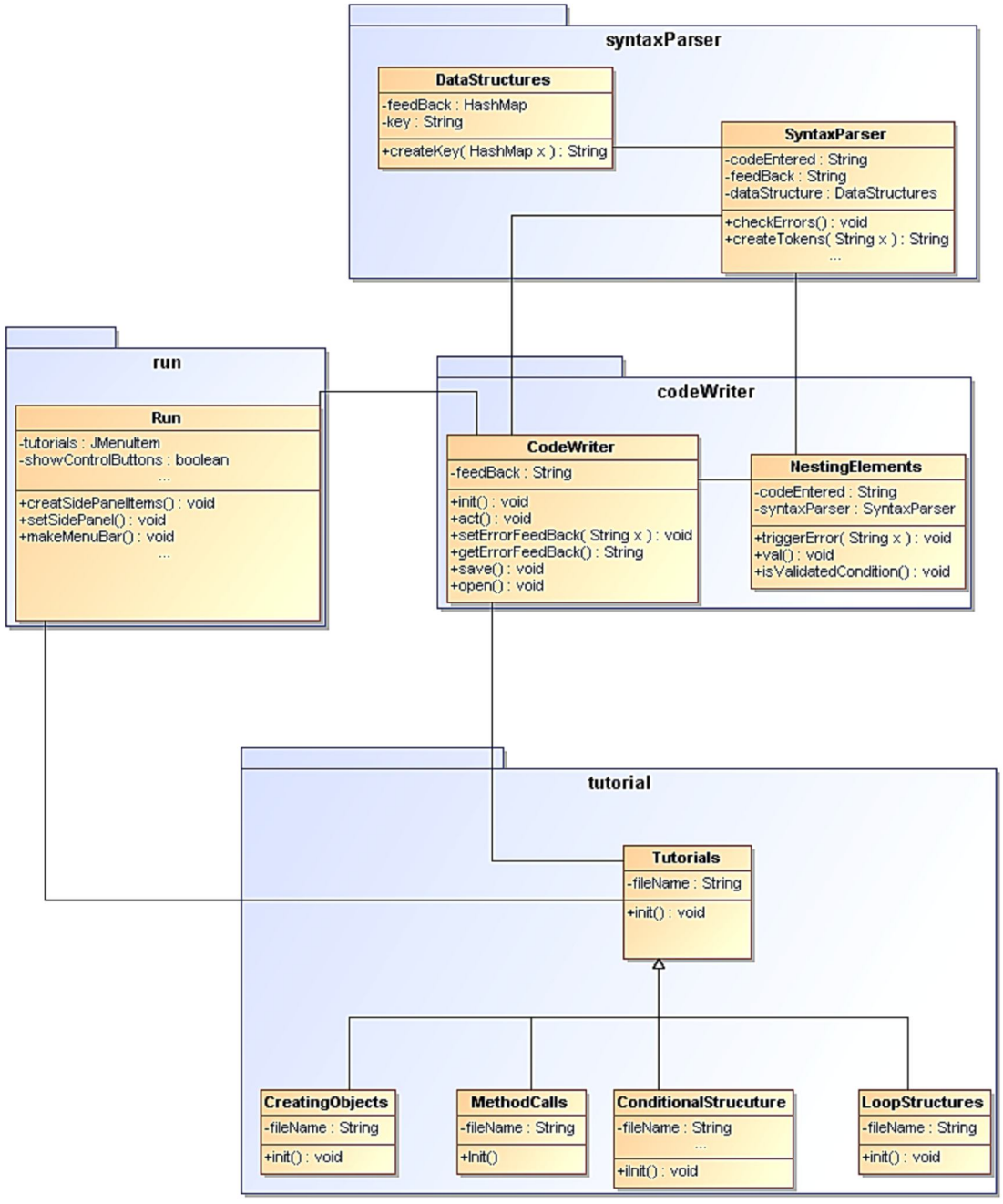
Feature	Undergraduate Project 2008/2009	New Robot World Simulator	Reason
F1	Switch between the Code Writer , control buttons and the location finder	Both displayed at the same time	Make the user see the code writer and Control buttons and choose the one to use
	Contains the location finder as the a menu item	Location finder replaced by the Status Bar	No need for a separate location finder menu as the status bar will be there
	Has no Status Bar	Has a Status Bar	Displays constantly the location of the selected cell
	Pop up when anomalies appear during robot movement	Provide pop ups and animations when anomalies appear during robot movement	Addition of animation to keep the user entertained
	The buttons and the code writer control object at different times	User able to control the same object using buttons and code writer interchangeably	User can start using the control at one time and switch to code writing on the same object
F4	No Interactive Code Writer	Has Interactive Code Writer	Code writer is seen from start when the Simulation is opened up
		Code Writer has a complex syntax checker	To allow the system to pick up errors suggest solutions
		Displays Java Codes output when students use Visualization control buttons	To allow the students to make relationship between java codes, human language and visualization
F5	Code Writer has a simple checker	Code Writer has a complex syntax checker	To allow the system to pick up errors and suggest solutions
		Can load contents from the Interactive Code Writer	To give users a starting point
F6	Output Java Statements	Output Pseudo codes	Close to human language and code

			writer already has Java statements
F7	Output Information	Displays the active cells	Always know where the student is in the visualizations
F8	Scans user inputs and alerts the user when he has made an error	Alerts the user where an error is made and provide assistance	Picks up errors and suggests corrections

Appendix B: Features that the New Robot World will contain that will be different from the Previous Robot World.



Appendix C: Iteration 1 Class Diagram



Appendix D: Iteration 2 class Diagram

Functional Requirement	**Test**	**Details**	**Expected**	**Actual**
FR 1	All the features are seen in Robot world window when the system loads the first time			
		Robot World	PASS	PASS
		Code Writer	PASS	PASS
		Control Buttons	PASS	PASS
		Text viewer	PASS	PASS
		Status Bar	PASS	PASS
		Menu Bar	PASS	PASS
FR2	Navigate through Control Buttons and Code writer		PASS	PASS
		Control buttons are clickable	PASS	PASS
		Write on the code writer	PASS	PASS
FR3	Visualize the robot world movements in 3D		PASS	PASS
FR4	Control the object by using control buttons		PASS	PASS
FR4.1		Create objects (robot, a cone and a barrier) by clicking on the control button	PASS	PASS
FR4.2		Move the robot forward by		

		clicking the control button move forward		
FR4.3		Move the robot backward by clicking the control button move backward	PASS	PASS
FR4.4		Robot turns left when the user clicks the button turn left	PASS	PASS
FR4.5		Robot turns right when the user clicks the button turn right	PASS	PASS
FR4.6		Robot picks a cone when the pick cone button is used	PASS	PASS
FR4.7		Robot drops a cone when the drop cone button is used	PASS	PASS
FR4.8		Robot moves into a cell where there a barrier when using the control button	PASS	PASS
FR4.9		Delete objects (robot, cone and barrier) from the Robot world by using control buttons	PASS	PASS
FR5	Manage objects by using Code Writer		PASS	PASS
FR5.1		Objects(robot, a cone and a barrier) are created by writing the java code to create objects on the code writer	PASS	PASS
FR5.2		Robot moves forward s by writing the java code to move forwards	PASS	PASS

FR5.3		Robot moves backwards by writing the java code to move backwards	PASS	PASS
FR5.4		Robot turns left by writing the java code to turn left	PASS	PASS
FR5.5		Robot turns right by writing the java code to turn right	PASS	PASS
FR5.6		Robot picks a cone when the pick cone java code is written	PASS	PASS
FR5.7		Robot drops a cone when the drop cone java code is written	PASS	PASS
FR5.8		Move a robot into a cell where there is a barrier when using the code written	PASS	PASS
FR5.9		Delete objects (robot, cone and barrier) from the Robot world by writing java codes on code writer	PASS	PASS
FR5.10		Use Conditional Structure to control object in Code Writer	PASS	PASS
FR5.11		Use Loop Structure to control object in Code Writer	PASS	PASS
FR6	Use control buttons and code writer simultaneously		PASS	PASS
		Create objects(Robot, Cones, and Barriers) by using Control buttons and manage them by using control buttons	PASS	PASS

		Create objects(Robot, Cones, and Barriers) by using code writer and manage them by using control buttons	PASS	PASS
FR7	Text viewer outputs Java codes when Control button are used		PASS	PASS
FR8	The Text viewer outputs pseudo codes when the code writer is used		PASS	PASS
FR9	Te status bar displays the location of the selected all the time		PASS	PASS
FR10	Menu bar contains a function that has Code Writer and Code Editor		PASS	PASS
FR10.1		The Interface changes to Code Editor when the sub-menu Code writer under menu Function is clicked	PASS	PASS
		The Interface changes to Code Writer with Control Buttons when submenu Code Window is chosen	PASS	PASS

Appendix E: Iteration 1 Test Cases

Functional Requirement	**Test**	**Expected**	**Actual**
FR11	The system detects where the error is in the piece of code entered by the user	Pass	Pass
FR11.1	The system detects Creation Statement Errors	Pass	Pass
FR11.2	The system detects object Method Errors	Pass	Pass
FR11.3	The system detects Conditional Structure Errors	Pass	Pass
FR11.4	The system detects Loop Structure Errors	Pass	Pass
FR11.5	Feedback given to users to show them where the errors are and how to rectify the error	Pass	Pass
FR11.6	Color the string of error code with Red Error	Pass	Pass
FR12.1	Introduction page describing the control structures and other object source of information of OO programming	Pass	Pass
FR12.2	User provided with Sequence Control Structure Tutorial	Pass	Pass
FR12.3	Users provided with Conditional Control Structure Tutorial	Pass	Pass
FR12.4	Users provided with Loop Control Structure	Pass	Pass
UR13	Sound to make sure the user knows the code has failed	Pass	Pass

Appendix F: Iteration 2 Test Cases

Structure	**Errors**
Object Creation Statement	The class name begins without Capital letter
	Class variable begins with Capital letters
	The keyword “new” is missed
	Unpaired brackets
	Cell value entered is not an integer between 0 - 5
	Color entered without type of color
	Missing semicolon
	Missing closing brackets
Methods Statements	The object variable begins with Capital letter
	Missing brackets and semi colon
	Space left between connected words
	The second word does not start with Capital letters
	Direction is not given to the robot
Conditional Structures	Invalid variable
	Variable starts with Capital letter
	Invalid Method
	Missing Opening brackets
	Missing closing brackets
	Missing Parameters brackets
	Missing brackets
	Method start with capital letter
While Structures	Missing Parameters brackets
	Method start with capital letter
	Missing Opening brackets
	Missing closing brackets
	Method start with capital letter

Appendix G: Errors that can be trapped by the syntax Checker

Appendix H: Anomalies where the system plays a sound

- Robot can not move forward
- Robot can not move backward
- Robot can not be added
- Barrier can not be added
- User has made an error when inputting java code

Feature	**Specific Suggestion**	**Status**
Code Writer	Change the color of the text displayed when user clicks the control button	Fixed
	Auto complete features	Left for future Improvements
Text Viewer	Change the way the active event is displayed	Fixed
Visualization Panel	Directions are not clear	Left for future Improvements
	The origin is different from normal Cartesian origin.	Left for future Improvements
	The cells should be interactive to students. Students should be able to click and use cells directly	Left for future Improvements
Language	Ambiguous use of Robot World and Simulator. Is robot word the whole system or is it just the visualization Panel	Improved
	Language is technical	Improved
	Instructions are not enough. Some feature is still not clear on what they do.	Improved
Miscellaneous	The load button on the Interactive interface is not needed	Fixed

Appendix I: Suggestion from Evaluators.