Low floor, high ceiling programming with App Inventor and the development of educational math-based games for Android

Vladislav Vekshin
B.A. (Mod.) Business and Computing
Final Year Project April 2013
Supervisor: Brendan Tangney

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I, the undersigned, declare that this work has not been previously submitted as an exercise for a degree at this or any other University

____________________  ________________
Name                   Date
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.
Acknowledgements

I would like to thank my supervisor, Brendan Tangney for his helpful suggestions, professional insight, and for keeping me on track throughout this project. I would also like to thank Barry Lynam for his suggestions the Number Line game, as well as Hugh Lardner who aided in the implementation of GPS movement game play mechanic. Finally, I would like to thank Kevin Sullivan as well as all of the students in Bridge 21 who were willing to take time out of their classes to help test the games.
# Table of Contents

Acknowledgements

Abstract

**Chapter 1 - Introduction** ................................................................. 1
  1.1 Low Floor, High Ceiling programming tools ......................................... 1
  1.2 Importance of Mathematics in Education ............................................. 1
  1.3 Maths-based mobile learning .............................................................. 2
  1.4 Report Overview .................................................................................... 2

**Chapter 2 - Background** ................................................................. 4
  2.1 Low-floor, high ceiling Construction Kits and Mindtools ....................... 4
  2.2 Related Work .......................................................................................... 5
  2.3 Affordances/Utilities of Mobile Devices ................................................. 5
  2.4 M-Learning through video games .......................................................... 6

**Chapter 3 - Low-floor, high-ceiling tools Overview** ............................. 7
  3.1 Logo ........................................................................................................ 7
  3.2 Lego Mindstorms NXT .......................................................................... 8
  3.3 Scratch ..................................................................................................... 9
    3.3.1 Limitations ...................................................................................... 10
  3.4 SLurtles .................................................................................................. 10
  3.5 Alice ......................................................................................................... 11
  3.6 Greenfoot ............................................................................................... 13
    3.7.1 App Inventor Features ................................................................. 16
    3.7.2 App Inventor Components ............................................................. 17
    3.7.3 Advanced User Functions ............................................................... 17
    3.7.4 Source Code Access ....................................................................... 18
    3.8.2 Common Themes ............................................................................ 20
    3.8.3 Exposure to programming concepts ............................................... 21
  3.9 Mindtool Summary .................................................................................. 22

**Chapter 4 - Design** ........................................................................... 23
  4.1 Development Environment ..................................................................... 23
  4.2 App Inventor Components ..................................................................... 23
  4.3 Components Utilised ............................................................................. 24
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.13 Discovered App Inventor Limitations</td>
<td>24</td>
</tr>
<tr>
<td>4.13.1 User Interface Design Limitations</td>
<td>25</td>
</tr>
<tr>
<td>4.13.2 Android User Interface Guidelines</td>
<td>26</td>
</tr>
<tr>
<td>4.13.3 Components required to be run locally</td>
<td>27</td>
</tr>
<tr>
<td>4.13.4 Blocks Editor Issues</td>
<td>27</td>
</tr>
<tr>
<td>4.13.5 MIT App Inventor Known Issues</td>
<td>27</td>
</tr>
<tr>
<td>4.3.1 Button</td>
<td>28</td>
</tr>
<tr>
<td>4.3.2 Canvas</td>
<td>28</td>
</tr>
<tr>
<td>4.3.3 Screen Arrangement</td>
<td>29</td>
</tr>
<tr>
<td>4.3.4 Clock</td>
<td>29</td>
</tr>
<tr>
<td>4.3.5 Sound</td>
<td>30</td>
</tr>
<tr>
<td>4.3.6 ImageSprite</td>
<td>30</td>
</tr>
<tr>
<td>4.3.7 Orientation Sensor</td>
<td>31</td>
</tr>
<tr>
<td>4.3.8 Location Sensor</td>
<td>31</td>
</tr>
<tr>
<td>4.4 Game Design Methodology</td>
<td>32</td>
</tr>
<tr>
<td>4.5 Math Wars Game</td>
<td>33</td>
</tr>
<tr>
<td>4.6 Number Line Game</td>
<td>33</td>
</tr>
<tr>
<td>4.7 Shared Screens</td>
<td>33</td>
</tr>
<tr>
<td>4.8 Points/Time System</td>
<td>34</td>
</tr>
<tr>
<td>4.10.1 Tilt (Orientation Sensor)</td>
<td>37</td>
</tr>
<tr>
<td>4.10.2 GPS (Location Sensor)</td>
<td>37</td>
</tr>
<tr>
<td>4.10.3 GPS Control Scheme - Gameplay Implications</td>
<td>37</td>
</tr>
<tr>
<td>4.11 Technical Challenges</td>
<td>38</td>
</tr>
<tr>
<td>4.12 Solutions to Challenges</td>
<td>38</td>
</tr>
<tr>
<td>4.13 Discovered App Inventor Limitations</td>
<td>39</td>
</tr>
<tr>
<td>4.13.1 User Interface Design Limitations</td>
<td>40</td>
</tr>
<tr>
<td>4.13.2 Android User Interface Guidelines</td>
<td>40</td>
</tr>
<tr>
<td>4.13.3 Components required to be run locally</td>
<td>40</td>
</tr>
<tr>
<td>4.13.4 Blocks Editor Issues</td>
<td>41</td>
</tr>
<tr>
<td>4.14 Design Summary</td>
<td>41</td>
</tr>
</tbody>
</table>
Chapter 5 - Evaluation ................................................................. 43
  5.1 Evaluation Methodology ......................................................... 43
  5.2 Testing on Students - Overview .............................................. 43
  5.3 Test Day 1 ........................................................................... 44
    5.3.1 Tilt control scheme ....................................................... 44
    5.3.2 Test Day 1 Feedback - SUS Results ............................... 44
    5.3.3 Test Day 1 Feedback - Questionnaire Results ............... 45
    5.3.4 Comments .................................................................. 45
  5.4 Test Day 2 ........................................................................... 45
    5.4.1 GPS/Tilt control scheme .............................................. 45
    5.4.2 Test Day 2 Feedback - SUS Results ............................... 46
    5.4.3 Test Day 2 Feedback - Questionnaire Results ............... 46
    5.4.4 Comments .................................................................. 46
  5.5 Accessing Feedback ............................................................. 47
  5.6 Evaluation Summary ............................................................ 48

Chapter 6 - Conclusion ............................................................... 49
  6.1 Results .............................................................................. 49
    6.1.1 Were Useful Educational Games Developed? .................. 49
    6.1.2 How effective is App Inventor as a low-floor, high ceiling tool? .... 49
  6.2 Criticisms ........................................................................ 50
    6.2.1 Testing Sample ............................................................ 50
    6.2.2 Mathematics Difficulty ................................................ 50
    6.2.3 Scope of Mathematics .................................................. 50
    6.2.4 Scope of Development .................................................. 51
  6.3 Future Work/Improvements .................................................. 51
    6.3.1 Expansion of Applications .......................................... 51
    6.3.2 How to Play Screen ..................................................... 52
    6.3.3 Combining the two applications .................................. 52

Bibliography ............................................................................ 53

Appendix .................................................................................... 57
Abstract

The core of this project can be divided into two congruent sections. The first section focuses on the constructionist learning model of low floor, high ceiling programming and the analysis of tools that embody this methodology, whereas the second is concerned with the development of educational math-based games for Android. The ideology of low floor and high ceiling programming stems from the desire to provide systems that are easy to use and accessible for novice users, while at the same time being powerful enough to develop interesting and challenging applications for the more experienced users. Through the utilisation of App Inventor to create math-based educational mobile games, this project aims to test the implicit constraints within App Inventor’s architecture in order to discover how low its floor really is, and how high its ceiling may currently be.

Smartphones provide the user all manner of affordances that other devices cannot. Portability, connectivity (SMS, Bluetooth, NFC, etc.), sensors (orientation and location), and recorders (audio, photographs, videos) are all powerful assets mobile phones now possess. For this project, smartphone sensors such as the orientation sensor and location sensor in particular, were of interest. Teaching and learning through mobile learning (m-learning) is another key driver behind this project. Considering mathematics plays a prominent role in Computer Science and is a subject that has seen a steady decline in recent years within the education system, the development of two maths-orientated video game applications within the m-learning domain became the area of interest; one that is also tied to the area of kinaesthetic learning.

The processes involved in this project included the first initial exploration and examination of App Inventor. Then, selecting the appropriate tools within the App Inventor environment and applying these tools towards the creation of the proposed maths based game applications within the App Inventor environment. Finally, the testing, evaluation and the outlining of any future improvements based on the findings resulting from the above. Through the evaluation of the two finished applications, App Inventor was discovered to be a capable low-floor, high ceiling programming tool for novice and advanced users alike.
Chapter 1 - Introduction

1.1 Low Floor, High Ceiling programming tools

“Low-floor (easy to use), and high-ceiling (powerfully expressive) are concepts central in the design of constructionist tools for learning” (Girvan, Tangney and Savage, 2012). App Inventor (MIT, 2013) along with other low floor, high ceiling programming tools such as Logo (Papert, 1980), Scratch (MIT, 2013), LEGO Mindstorms NXT (LEGO, 2013), SLurtles (Girvan, Tangney and Savage, 2012), Alice (Alice, 2013) and Greenfoot (Greenfoot, 2013) have empowered a whole new age group consisting of primary and secondary school students to create applications for the Android platform - a task that had not been possible previously due to the numerous demands and requirements of the Android development process. As such, the tools listed above are considered to be low floor, high ceiling development environments. With these types of programming environments, many technical barriers are removed, and the entry requirements to use are heavily alleviated. This inherent nature of low floor, high ceiling tools opens up a whole host of opportunities for potential developers from all backgrounds and varying levels of programming experience.

1.2 Importance of Mathematics in Education

Mathematics was chosen as the underlying research topic for this project for several reasons. First, it is one of the core subjects of both the current junior and leaving certificate syllabus and is therefore an instrumental facet of a child's early life. Second, a certain level of mathematical understanding is required of students wishing to pursue the field of Computer Science and is therefore of intrinsic value for future students in this field, with universities and academics also holding an interest in their success as university places have to be filled. Third, the grade level of mathematics within secondary schools has been deteriorating over the last number of years, and while there have been numerous attempts to rectify this problem with initiatives such as Project Maths (NCCA, 2013), they have as of yet done little in the way of mitigating the problem. The potential implications of smartphones within education and learning are still
being discovered and are constantly being developed through research and teaching. Thus, the creation of maths-based educational games in aid of the current mathematical education problems was proposed for this project.

### 1.3 Maths-based mobile learning

Creating fun and interactive maths games can potentially improve or contribute to a student’s learning experience (Spikol and Milrad, 2008). With the addition of tilt, and GPS movement control mechanics, the possibility for further student engagement and interaction through m-learning by playing the games with such control schemes increases (assuming they do not detract from the game play experience). Placing the player in non-static, always moving gaming environment implies that not only do the children have a chance to put into practice the mathematical skills they learn in class, but they also get an opportunity to learn and gain experience through other forms of learning; kinaesthetic and tactile learning methods specifically.

The aim of this project was to develop mathematically inclined mobile games that could potentially be used to engage children as well as aid with their mathematical learning in and out of the classroom environment with the potential of improving their problem solving abilities. The games were designed to have a strong focus on kinaesthetic learning by applying a multitude of smartphone sensors, and as an attempt to push the limits of App Inventor while attempting to create an exciting and engaging learning experience for children.

### 1.4 Report Overview

Chapter 2 will begin by discussing the background of low-floor, high ceiling programming tools, and will then go through any related work and motivations behind this project, with mobile learning and an outlay of the affordances and utilities mobile devices present us in this day and age.

Chapter 3 will contain a general overview of the programming construction tools mentioned earlier, with an in-depth investigation into App Inventor and its usefulness as a low-floor, high ceiling environment will be provided.
In Chapter 4 a step by step overview of the design of the two proposed maths applications, Maths Wars, and the Number Line Game with an outline and review of the design decisions and challenges faced throughout the project implementation process will be discussed. Subsequently, the challenges, limitations and the “ceilings” of App Inventor that were discovered throughout the development process will be outlined.

A detailed evaluation of the project will then be undertaken in Chapter 5. Firstly, the feedback received the intended users will be discussed and analysed, and secondly, an analysis of the project as a whole will then take place, taking in the feedback produced which will focus on both the educational merit and overall quality of the two maths games as a whole.

Finally Chapter 6 will conclude in a brief outline for future work and possible additions/extensions that could potentially be made based on the testing and feedback collected.
Chapter 2 - Background

2.1 Low-floor, high ceiling Construction Kits and Mindtools

The fundamental idea of “learning through designing” is the belief is that that the best learning experiences, for most, come when they are actively engaged in designing and creating things, especially things that are meaningful to them or others around them (Resnick and Silverman, 2007). This is the core ideology and foundation of the constructionist model in learning and is central to all manner of applications such as Scratch, Logo, Alice, Greenfoot, App Inventor and many others.

The overarching goals and the motivations behind the creation of the above tools as “construction kits”, as well as the encouragement of constructionist approaches to learning are: to help children become more fluent and expressive with new and existing technologies; to help them explore important concepts in the domains of mathematics, science, and engineering through their expressive activities; and, most broadly, to help them become better learners (Resnick and Silverman, 2007).

We are now living in an ever-changing knowledge society where we must constantly adapt to new information and continually respond and find solutions to unexpected problems. Through a science like programming and its inherent problem solving characteristics, we can help the younger generations by giving them the opportunity to create and think creatively and thrive in this new Creative Society (Resnick, 2007). The concept and use of Mindtools enhances and propels such a society. A Mindtool, as described by Jonassen (1999), is a computer-based tool or learning environment that engages learners in constructive, higher-order, critical thinking about the subjects they are studying’. Modern Mindtools such as Scratch, Greenfoot, Alice, Lego Mindstorms, and App Inventor are seen as “construction kits” that engage children in the art of design and creation.
2.2 Related Work in TCD

Previous research on the ability to control mobile games through a smartphone’s Location Sensor has been conducted. A draughts (or checkers) game was an augmented reality game built with App Inventor. In this game, players could use their phones to move around like human checker pieces on a large outdoor field. The game tracked movements using GPS positioning, and showed the location of each checker on the virtual board depicted on the players’ screens. The draughts game was programmed by Trinity College Dublin undergraduate Hugh Lander (MIT, 2013). Another game was developed by Hugh that allowed for a multiplayer game of Pong to be played with both on-screen and GPS movement controls (MIT, 2013). Both of these applications served as the foundation and inspiration for this project.

One of the proposed games for this project (Math Wars) was influenced by a paper on the exploring the potential of modern mobile games (O’Byrne, 2011). O’Byrne explored video games and the topic of game-based learning and attempted to create a set of engaging maths-based games that utilised modern smartphone touch screen and sensor technology. One of the games he developed in IOS was the inspiration for this project’s game, Math Wars. The imbuing of a control scheme that utilises a smartphone’s Location Sensor for movement with a similar type of game to the above was an attempt to build on previous research.

2.3 Affordances/Utilities of Mobile Devices

The wide adoption of smartphones is an ever-growing phenomenon in the current mobile phone market. They have become something more than simple playthings and tools towards a singular purpose. Today, smartphones can be described as multi-functional, all-purpose devices that combine the portability of a mobile phone with some of the functions of a desktop computer. They also provide an interesting array of sensors that are outside the limitations of PCs. Modern smartphones come equipped with powerful computational technology, complete with a multitude of useful sensors that can be used for a wide variety of tasks for practical uses such as navigation, for the measurement of angular distance, tilt, vibration, orientation and many more.
For this project, the sensors that were of interest were the orientation sensor for Tilt control using the pitch and roll readings for movement, the azimuth for compass and directional awareness, and the location sensor for the GPS coordinates obtained through the latitude and longitude readings.

2.4 M-Learning through video games

Over the past few years there has been a wealth of research carried out worldwide into the educational properties and merits of computer games, much of which has been positive. Several studies have validated the use of digital games as learning tools (Facer, 2003; McFarlane et al., 2002). These studies show that educational games are powerful educational tools, that when properly used, motivate and stimulate learners towards self-discovery, personal research, exploration, and imagination. Rosas et al. (2003) proposed that utilising digital games in educational instruction positively influences learning achievement, development of cognitive ability, learning motivation, and concentration in learning.

M-learning was a constant and valuable imperative for this project; both utilizing and empowered by kinaesthetic learning. M-learning can help develop and prepare students to become knowledge workers towards a smart economy by combining learning through subjects such as mathematics and the utilities that a modern smartphone provides. The intention is to generate an invaluable learning process and engages students with mathematical problems in a practical environment. With today’s children being exposed to technologies such as touch screens and mobile sensors, it provides them with a pre-established familiarisation with smartphones. By choosing to explore the affordances and capabilities a smartphone provides, the mobile video game medium became an evident choice for teaching mathematical concepts to children.
Chapter 3 - Low-floor, high-ceiling tools
Overview

3.1 Logo

Among the various applications being taught to school children is programming with the computer language, Logo. It was the seminal programming construction kit to be released from within MIT, developed by Papert (1980) who was working with the famous psychologist, Piaget at the time (Logo Foundation, 2011). Just like other programming construction kits, Logo promotes learning by discovery rather than by instruction. Logo was Papert’s vision to what would become a tool that empowered children and provided them with a graphical environment rich in opportunities for problem solving.

By playing with a turtle (usually a visual triangular object) and writing procedures for the turtle to move, it enabled developers to draw and move shapes on a screen. In this way, children developed systematic strategies for problem solving and began to think in a programming-oriented way. As experience accumulated, the systematic strategies gave way to an intuitive art of problem solving and the understanding of concepts of geometry and Logo programming. The psychosocial effects of such experiences were that children developed a positive sense of self and good attitudes towards learning. In Papert’s own words, “children would become epistemologists” (1980), that is, they would learn how to learn and become aware of their own thinking processes.

![Logo Design View](image_url)

Figure 1 – Logo Design View
3.2 Lego Mindstorms NXT

Released in 2006 (Kim and Jeon, 2008), LEGO Mindstorms NXT enables the learner to design, build, and programme a robot through a programmable brick that has the ability to control motors and sensors. The sensors (light and touch) allow the robot to interact with its environment in accordance to the programme that has been designed. The programming interface can be installed on a PC, can then be transferred directly to the brick through an infrared connection. Mindstorms NXT is another block-based development environment that enables the use and manipulation of all of the basic programming concepts without any knowledge of the syntax (Savage et al., 2003).

While there is some variety, for example in length and width, the form of blocks available is typically limited to cuboid shapes (Figure 6), although others are available. With the provided tools, users can decide port, direction, steering, power, duration, and the robot’s next action. Every block has an appropriate set of parameters with adjustable values (e.g., “turn left wheel 15 rotations”). A program can be developed through a sequence of software instructions determined by placed blocks down into the design view. These instructions then interact with the internal and external state of the robot, performing calculations using this input, and initiating actions by commanding the actuators, thus changing the internal and external state (Slange, Keulen and Gravemeijer, 2010).

Figure 2 – Lego Mindstorms NXT Software
3.3 Scratch

The Scratch project began in 2003, and the Scratch software and Web site were publicly launched in 2007 (Maloney et al., 2010). Scratch is a block-based environment for content creation. Scratch scripts are built mainly by dragging and dropping brick components onto a design window. The Scratch user interface strives to make navigation easy. It uses a single window, multi-panel design to ensure that key components are always visible. Scratch avoids floating palettes, which can get buried, and minimizes the use of panes that show only on demand.

The command palette on the far left (Figure 7) is always visible. The commands are divided into eight categories such as Motion, Looks, Sound, and Control. This avoids long, potentially overwhelming lists of commands. In most palettes, all the commands can be viewed without scrolling and in each category, the most self-explanatory and useful commands appear near the top of the command palette. Command blocks are also colour-coded by category, helping users find related blocks (Maloney et al., 2010).

![Scratch Design View](image)

Figure 3 – Scratch Design View

A key feature of Scratch is that it is always live (Maloney and Smith 1995). There is no compilation step or edit/run mode distinction. Developers can click on any available command to see what it does. Developers can even change parameters
or add blocks to a script while it is running. By eliminating potentially jarring mode switches and compilation pauses, Scratch helps users with testing debugging, and adding onto their projects.

### 3.3.1 Limitations

However, Scratch does have some evident limitations: no user-defined blocks, no return values, no file interaction or management, no network interaction, no dynamic object creation, the program cannot draw on top of *Sprite* Images (only on the background), no string variables or any real string handling. It is a great environment for learning to think creatively within its constraints, but developers can hit its limits pretty rapidly. Older children have found that Scratch is quite limited as a construction tool (Kerr, 2008).

### 3.4 SLurtles

Borrowing design concepts from Lego and Turtle graphics, SLurtles build upon (Scratch for Second Life) S4SL and are designed to provide learners in Second Life with programmable tools with which to create personally meaningful and shareable artefacts (Girvan, Brendan & Savage, 2012). S4SL is built upon the Scratch framework and provides a low-floor, visual programming interface for Second Life (Rosenbaum, 2008). In the same way, graphical command blocks are selected and dragged to the scripting area where they are snapped together to create a programme. SLurtles came about due to the inbuilt tools of virtual worlds such as Second Life and Active Worlds presenting the novice with a high-floor (steep learning curve) barrier to overcome (Girvan, Brendan & Savage, 2012; Dickey, 2005). Borrowing design concepts from Lego and Turtle graphics, SLurtles are designed to provide learners in Second Life with programmable tools with which to create 3D objects in a virtual world (Girvan, Brendan & Savage, 2012).

S4SL interacts with Second Life through a block-based Scratch interface which S4SL was developed from. On the click of the ‘Copy Linden Script’ button in the user interface, the S4SL code is compiled into equivalent Linden Scripting
Language (LSL) code. The user then returns to Second Life, creates a default script, pastes the code and places it in an object.

To draw 3D objects through the S4SL interface, the user must use a Line Segment object and its inherent pen down command. By using pen down and following with a movement in any direction, an instance of the 3D Line Segment is drawn within Second Life. The location and length of the new object are determined by the location of the parent object when the pen down command was issued and how far it travelled in one action (Girvan, Brendan & Savage, 2012).

![Image of Scratch and Second Life interface]

**Figure 4 – Scratch for Second Life Design View**

### 3.5 Alice

Alice, from Carnegie Mellon University, is a 3D programming interface developed to attract more students to the computing field. It is a drag and drop environment which does not require developers to write textual code. Previous studies have shown that Alice is a suitable interface to be taught in introductory programming courses (Cooper, Dann and Pausch, 2000). Syntactically, there is no actual free-text typing so that clearly younger children can mechanically handle Alice, as early as the age of 10 (Utting et al, 2010)
Alice is a 3D interactive animation program visualization environment. Novice programmers build animated 3D movies and author games as they learn introductory object oriented programming concepts. The software as well as the source code is freely available from the Alice website, and instructional materials are freely available to teachers also.

Alice offers two significant benefits for teaching introductory programming (Gavan and Anderson, 2012). The code is presented in a style similar to Java, which enables students to focus on enhancing their problem solving skills without the necessity of learning language syntax simultaneously. As the code presented looks like Java code, users become aware of the syntax such as the use of blocks and semi-colons. The second key advantage of using Alice is based in its adoption of an object-oriented mentality. Students place objects into a virtual world and interact with those objects in the way of sending messages to the object concerned by selecting the appropriate class from the Alice Class Library. The visualization of these concepts has proven extremely useful when introducing object-orientation to students.

It is not possible to enter an invalid statement because of the drag and drop validation system inherent in Alice. Parameters can also be added or changed through block’s context menus, but the structure of the statements themselves cannot be changed, nor can their labels or captions. The drag and drop nature of
Alice implies that it is relatively easy to insert new blocks, but it is also more tedious to rearrange or replace blocks once they are in place (McKay, 2012).

### 3.6 Greenfoot

Greenfoot (Henriksen and Kolling, 2004) is a Java-based system developed at Kent, used for teaching object oriented programming through games. In Greenfoot, the programmer enters code in a colour-highlighted text editor, similar to what is found in many IDE’s. Its primary purpose is to teach students from about 14 years of age and onwards, and is also aimed at students in university education that may need help with programming (Kolling, 2010). The editor uses subtle background colours to highlight structure, putting boxes around statements, loops and other constructs. It also highlights keywords.

![Greenfoot Design View](image)

**Figure 6 – Greenfoot Design View**

When a developer opens Greenfoot, they are greeted with a main window showing the Greenfoot World (the area where the program executes). Greenfoot has 2 persistent super classes that are always present, the World and Actor classes. Below the world view there are a variety of execution controls that allow the running of, or stepping through of the program.

Greenfoot internally uses the standard Java compiler, along with the standard Java Virtual Machine (Something App Inventor does not). This allows for the
coding of simple, one line methods by right clicking on a block and accessing its source code. However, despite the language being in Java code, it is supported in simpler ways so that the novice user can still be able to create. There is no need for a Main class, or indeed the need to write test methods. As soon as the World and Actor(s) compile, the developer may call the Act method (which is inherent from the Actor super class) in order to invoke the desired action method once. There is also the ability to run the method continuously by clicking the Run button. Greenfoot gives the user all of the graphical tools from the offset, and this leaves the developer to focus entirely upon the logic and logical behaviours of the Actors as Greenfoot takes care of the graphics and graphical animations of the actors implicitly (Kolling, 2010).

Figure 7 – Greenfoot IDE Screen

Greenfoot is divided into five classes and its API can be summarised in roughly two letter-sized pages. Students can therefore learn very quickly the in-built methods available to them, and the documentation is readily available within the Greenfoot environment.

3.7 App Inventor

App Inventor was first announced as a small project of Google Lab in late 2010, and has since been transferred to MIT Mobile Learning Centre after Google terminated App Inventor in December of 2011 for public use under the spirit of open source (App Inventor.mit.edu). The authors of O’Reilly Books’ App
Inventor describe the program as “part of an on-going movement in computers and education that began with the work of Seymour Papert and the MIT Logo Group in the 1960s, and whose influence persists today through many activities and programs designed to support computational thinking.” (Michael Castelluccio, 2012).

App Inventor is a web application tool that provides an easy to use working environment towards the creation of mobile applications for the Android operating system, and is targeted at anyone who is unfamiliar with, or has a basic understanding of computer programming. App Inventor provides a graphical and online programming environment that lets users create Android applications through a two-stage design process. The developer begins by designing their user interface and dragging any Components (buttons, text boxes, images etc.) they may require within the application Designer (which is run in a web browser and hosted by Google’s App Inventor server).

![App Inventor Designer](image)

Figure 8 – App Inventor Designer

Then, developers must define the program logic and functionality of these components through the Blocks Editor which requires the Java applet to run. Here, App Inventor provides code snippets that are pre-packaged within coloured blocks and have extensions with openings that work somewhat like a jigsaw puzzle. This is essentially brackets sand parenthesis if compared to high level programming languages. With these jigsaw pieces provided, an App
Inventor application creator can snap together the appropriate pieces to build routines and methods.

The Designer and Blocks Editor communicate together to create complete and finished applications that are then available for download in Android’s standard package format (.apk) exclusively for the Android Operating System. Within the App Inventor Blocks Editor, the developer is able to drag and drop predefined in-built features and components with the added ability of being able to define variables of numerical, text and Boolean values as well as creating their own methods with or without parameters or return values depending on the required end result.

![App Inventor Blocks Editor](image)

Figure 9 – App Inventor Blocks Editor

### 3.7.1 App Inventor Features

App Inventor allows you to build mobile games and applications for smartphones. This is distinctly different from Alice, Scratch, Logo and the other development environments as they only allow for desktop applications or for the manipulation of robots (LEGO Mindstorms NXT). Saying this, App Inventor is limited to Android development so it really depends on what platform the user wishes to build applications on. For the novice, desktop applications are significantly easier to set up and run despite App Inventor providing an in-built Android emulator which forgoes the need (in most cases) for a dedicated Android device.
App Inventor has the useful feature of being able to debug applications in real-time on an Android device, enabling the developer to view, edit and test programs on the fly. The ability to debug in real-time proved extremely useful for this project and was utilized extensively throughout. App Inventor also allows the developer to use a built-in emulator that emulates the Android environment directly through the *Block Editor*. Another important thing to note is that App Inventor is supported on Linux, Windows and Mac OSX. Cross platform support is important as it allows developers to work on their projects wherever they go, on whichever OS they prefer or have access to. App Inventor has great flexibility in this regard.

### 3.7.2 App Inventor Components

App Inventor equips developers with a wide variety of tools necessary in building competent Android applications. In-built features or *Components* (as they are known as in App Inventor) come under headings such as Basic, Media, Animation, Social, Sensors and Screen Arrangement. App Inventor succeeds in sheer functionality providing a wealth of tools that can access almost all of a smartphone’s inherent features and functions. In this regard, it does indeed provide new users with an easy to use construction kit, while giving more advanced users enough to keep them occupied and experimenting with the various components provided. The tutorials provided on the main MIT App Inventor website serve as a starting point and guideline for the development potential of App Inventor. The tutorials empower developers with a basic, as well as more advanced knowledge for those unfamiliar with the App Inventor environment.

### 3.7.3 Advanced User Functions

App Inventor also provides optional advanced user functions that developers more familiar with the App Inventor environment, may utilise. By left clicking anywhere in the *Blocks Editor* workspace, a quick access menu pops up that features all of the in-built functions that can be generated as blocks without having to go through the blocks list on the left hand side of the editor. It does not support user generated variables and methods however so the developer has to either find these through the blocks list or type them out manually.
The App Inventor Blocks Editor also supports power user functions such as being able to type blocks into the editor instead of just dragging and dropping from the left pane. By left clicking within the blocks editor screen, the developer can type and search for numerical functions, in-built functions, and user created variables as an alternative to finding them through the navigable menus. This allows for increased coding speed that is similar to text based coding in java, although in a much more simplified and restrictive manner. Additions such as these reinforce the fact that App Inventor was built to have added functionality that an advanced developer would benefit from.

### 3.7.4 Source Code Access

App Inventor also provides the ability to access and download the entire source code library through the App Inventor releases website (App Inventor, 2012). Despite the source not being updated since February of 2012, as well as being an earlier version than the one currently held by the App Inventor web application proper, it allows for a developer to make changes to the blocks code by way of the Java programming language, and running a version of the App Inventor application locally. There is currently little supporting documentation, and MIT are not accepting contributions to the code right now, and because of this it is not easy to get up and running. A Linux operating system such as Ubuntu is recommended for source code manipulation. There is basic support provided for Windows and Mac as well, but there is even less information and documentation available in order to accomplish this.

### 3.8 Comparing Mindtools

#### 3.8.1 Target Age Groups/Ease of Use

Scratch, App Inventor, SLurtles, LEGO Mindstorms NXT and Alice all implement a drag and drop, block-based programming approach whereas Logo and Greenfoot require almost all functions to be typed. Logo does provide an easy to learn set of basic instructions, however is hindered by it lacking a high-level programming language foundation that Greenfoot provides with Java. Overall it is easier to get to grips with Scratch, followed by App Inventor, LEGO Mindstorms, and Alice and for novice developers.
In Greenfoot for example, standard Java syntax is used. That alone imposes a fairly hard maturity boundary for kids, and implies that the accessibility floor is much higher compared to other comparable development environments. Due to inherent complications that come from textual programming such as syntax errors, it would be very difficult for children under the age of 14 to understand where they are going wrong (Utting et al., 2010).

In Scratch, the contrary is true. The main goal is to “lower the floor” for programming (Utting et al., 2010). The aim is to get children started with programming as soon as they become interested in the concept of creation and the ability to make things work and do what they want them to do. Therefore Scratch is suited for children ages 8+.

Alice is suited primarily for students who are beginning to come across more abstract problems in mathematics such as algebra, and are in the process of getting to grips with more advanced mathematics (ages 13+). Such understanding gives them a sufficient background to be able to tackle a more rigorous approach to problem solving. Alice has also frequently been taught to college students in non-majors courses, which has more often than not experienced great success (Cooper, Dann, and Pausch 2010).

On the other hand, App Inventor targets an age group that lies somewhere in between Scratch and Alice. However as outlined previously, and due to the more complex setup procedure to get App Inventor up and running prior to use, it requires a little more know how than Scratch. The separation of the Designer and the Blocks Editor may also be somewhat jarring to the uneducated user. In this regard, Scratch has a lower floor than App Inventor and an age of 12 years and over would be recommended.

With all Mindtools except App Inventor, the developer does not have to worry about having a constant internet connection. They are desktop application that developers are able to download and install directly onto a PC. These applications are relatively easy to set up, and require far fewer steps to open and to get started.
creating games compared to a web application such as App Inventor, where multiple steps are required to access both the *Designer* and *Blocks Editor*. Instead of providing the user with two views (*Designer* and *Blocks Editor*) as in App Inventor, most comparative Mindtools simplify the design and programming process by only giving the developer one view. The developer therefore does not have to worry about installing third party applications (Java JDK) or any other additional file packages.

### 3.8.2 Common Themes

Users of Mindtools require a tool that is high level enough to do something useful, but low-level enough to allow flexibility, and to allow for the composing of their own actions; the inherent definition of the low floor, high ceiling ideology. A number of important design goals underpin the interaction design any of the observed Mindtools. At the highest level, they can be summarized in two points, from two different perspectives (Kolling, 2010):

1. From the student’s perspective, the goal is to make programming engaging, creative and satisfying.
2. From the teacher’s perspective, the goal is for the environment to actively help in teaching important, universal programming concepts.

Engaging and empowering the user is the most important theme and is the underlying goal for all of the mentioned Mindtools. All systems strive to connect the user with their interests depending on the context e.g. games, stories or simulations. This is a contrast to conventional programming where less interesting functions to the user are commonly utilized and required of the developer; for example the generation of prime numbers or abstract mathematical concepts (Utting et al., 2010). There is also a strong focus on problem solving within each environment, just like in any text-based programming language. The intention is to teach the developers to identify errors and then find ways around them and solve them through their own ingenuity.

Within Scratch, the main focus is on quickly and easily achieving something interesting by moving and animating objects on the screen, forgoing the teaching
of more advanced programming concepts in favour of simplicity and ease of use. Within others, such as App Inventor and Alice, the focus isn’t entirely on instant ability to produce content and playing within the development environment. Tutorials and other sources are recommended if wishing to develop something worthwhile.

All systems strive to eliminate the inherent difficulties associated with textual programming, although in contrasting ways to one another. For instance, Mindtools such as Alice, Scratch and App Inventor remove syntax issues due to their “block based” nature which allows the user to focus on the logic behind the program. Greenfoot provides the user with a “main” method as well as a tiny yet powerful class library that distances the user from the overwhelming large and complex set of Java APIs. The Actor/World framework makes it possible to start by writing short, self-contained methods instead of creating and organising custom-built methods (JM, 2010).

3.8.3 Exposure to programming concepts

One major aspect of teaching programming (specifically object oriented) that is lacking within App Inventor, Scratch, LEGO Mindstorms NXT, and in a lesser respect Alice are the teaching of fundamental concepts of object orientated languages such as Java and C++. Below is an outline of the fundamental object oriented programming concepts (ref):

- A program consists of a set of classes
- From these classes, objects can be created
- All objects of the same class offer the same methods and hold the same fields
- Each object holds its own values for its fields and has a unique state
- Developers can communicate with objects by calling one of their methods
- Methods may have parameters and they also may return values.
- Parameters and return values have types

All of these concepts are traditionally difficult to teach, but are basics each budding programming must know when building applications through object oriented languages. One of Greenfoot’s strengths is that is can move past the
abstractness of these concepts as presented through traditional teaching methods, and also provide a visual learning experience by guiding users through interactions with such concepts in a more apparent transparent. Something the other development environments find themselves lacking in.

With an environment such as Greenfoot which is explicitly built around Java, it is apparent as to how each Actor functions by accessing the context menu and “inspecting” that Actor. Here, the Actor’s relationships with the Object and Actor classes are explicitly apparent and makes it easy for the developer to see exactly how a class hierarchy is structured and visibly teaching them the fundamental concepts outlined above. The developer also has the ability to construct user-defined methods; users get to build their own commands by decomposing and adding onto existing ones. It is easy to animate actors on the screen, but it would take longer for novice users to get the hang of using textual commands rather than blocks (SC, 2010).

3.9 Mindtool Summary

Scratch tries to minimize complexity and lets its users learn entirely through play as soon as they start working within its environment. SLurtles follows the same principles but can be observed being somewhat more complex as it adds another dimension into the environment (3D). App Inventor follows in the footsteps of Scratch, but also provides the developer a more complex suite of tools to experiment with. Greenfoot plays a little more of a teacher role and tries to guide the user into what it wants to show then. Alice is somewhere in the middle, being powerful enough to produce competent applications as well as giving developers the opportunity to increase the complexity of their applications.

This very directly reflects the different stages of learning of the various targeted age groups and shows that all of these Mindtools share a common philosophy: Let the user play first, then achieve something, allow for creativity, and then sneak explanations about what is going on when you’re working with the system along the way. Thus, a sequence from Scratch to Greenfoot or from Alice/App Inventor to Greenfoot can work well and provide a progressively more realistic programming experience (Utting et al., 2010).
Chapter 4 - Design

From the outset, the design of the two games was centred on the utilisation of App Inventor to create competent and intriguing math-based educational mobile games, while attempting to evaluate App Inventor as a low-floor, high ceiling construction tool. Below, the tools and components utilised throughout development will be discussed, and the two games (Math Wars and Number Line) will be outlined in detail.

4.1 Development Environment

Most Android applications are developed using the Java programming language combined with the Android Software Development Kit (SDK). In order to start writing code a Java Integrated Developer Environment (IDE) is required. There are a variety of Java IDEs available such as Eclipse, Netbeans and JCreator. For this project however, App Inventor alone was used as the development tool; one that cuts through much of the complexity of Android software development.

The way App Inventor manages to circumvent text based programming and the need for database management skills is with a graphical block programming system called OpenBlocks and an easy-to-use database component, TinyDB (Michael Castelluccio, 2012). The TinyDB is a Designer component that comes built into App Inventor and bestows on the developer the ability to store persistent data that remains in the application’s cache memory even when an application is closed.

4.2 App Inventor Components

App Inventor functions around a set of pre-defined core Components within the Designer which allow the developer to create and manage how their applications will look and feel. These Components are fundamental to the building of applications in App Inventor, and they enable the developer to alter logical behaviours as they see fit within the Blocks Editor.

App Inventor offers a multitude of components to the developer that range from Basic components like the Button, Label, Clock and the Canvas that would be
utilized in almost every application, to more specific ones such as being able to access the phone’s camera, contacts, texting, web HTTP and many more. App Inventor also contains a series of components on offer that interface with Lego Mindstorms NXT and allow for the interacting and controlling of a Lego Mindstorms robot’s functions through the Bluetooth component.

For this project in particular, the following App Inventor components proved valuable for the task of developing video games. Below is a detailed outline of each component, and how they were used in the creation of the Math Wars and Number Line games:

4.3 Components Utilised

4.3.1 Button

These are perhaps the simplest way in which a game developer can get input from the game player. The button component was used for menu navigation, and the selection of maths problems in the Math Selection Screen as toggles (to diagram). App Inventor allows the developer to upload their own custom images and use them as backgrounds, icons, and buttons instead of the pre-sets provided in App Inventor. Custom icons and backgrounds were used in both games to give them a slightly different aesthetic to each other, while retaining a somewhat similar and consistent theme between the two.

4.3.2 Canvas

The Canvas component is useful for multiple reasons. First, it is the component where game objects are drawn. Shapes such as rectangles, circles, and custom Sprites can be drawn on these by using the methods provided by App Inventor within the Blocks Editor. Second, they are containers for other components known as Sprites (detailed below), and provide a set of bounds within which Sprite objects can be drawn and moved around in the Canvas. Third, Canvas objects can be used as a source of touch input. The Blocks Editor supports both "dragged" and "touched" events that the Canvas can respond to.

Fourth, these objects can be used as a spacing mechanism in App Inventor for screen layouts. E.g. if a developer wishes to centre a button on the screen, he/she
can add an empty Canvas or Label on each side of a Horizontal Layout (or vertical), and set the width property of each Canvas to fill parent. The end result will be the centring of the button or whatever other object is between the two outside Canvases. If however, the developer wishes to move a component like a Button to a certain position on the screen, it requires a little more tweaking and trial and error technique.

4.3.3 Screen Arrangement
By using items under the Screen Arrangement heading, one can group a series of components together into an arrangement. This is necessary if the developer may want to have a set of components in a line, underneath each other, or in a grid style. Again, to manipulate the positioning of components, a little experimenting is required.

4.3.4 Clock
The Clock or Timer component is instrumental for game development as it allows for the enforcing of time limits on the player. Time limits motivate players to concentrate on the task(s) at hand and attain game goals in as sufficient manner as possible. In App Inventor, the clock component contains an “Interval” property that the game developer can define and modify at any time. The Clock component fires an event after each interval expires and so a game developer can easily react to the firing of these clock timer events in the Blocks Editor in order to perform certain tasks (e.g. updating the location of a Sprite component) every few milliseconds.

4.3.5 Sound
The Sound component allows for the addition of sound effects within the App Inventor environment. These are easy to add in App Inventor and are uploaded and stored the same way as image files. The Sound component is an invisible component in the Designer that allows the developer to select a source sound file and to play this file in response to other actions from the Blocks Editor. Sound was extensively utilized in both of the games and functioned as indicators for when the player was correct, incorrect or in the event of a Game Over.
4.3.6 Image Sprite

Sprites are a fundamental part of any game that requires animation or motion. They are components that can be visualized by any image that the developer can upload to the project. Animated images like GIFs however, are not supported by App Inventor. The ImageSprite component supports rectangle-based collision detection and is restricted to movement within the bounds of the Canvas in which it resides. This implies that any moving objects in a game cannot leave the restrictions of a Canvas component, and can only interact with other objects/components within that same Canvas component.

4.3.7 Orientation Sensor

The orientation sensor on a smartphone is used to determine the phone's spatial orientation. In App Inventor, the orientation sensor is a non-visible Component that the developer is able to add to a project within the Designer. It tracks movement in three values, pitch, roll and azimuth, and can report these values in degrees.

**Roll:** The Orientation sensor's roll reading was necessary towards the control of the Tilt game play mechanic and allowed for the movement of the ball sprite in the Math Wars game from left to right. The Orientation sensor’s roll value reads 0 degrees when the device is level, increasing to 90 degrees as the device is tilted up onto its left side, and decreasing to −90 degrees when the device is tilted up onto its right side (Android, 2013).

**Pitch:** The Orientation sensor's roll reading was necessary towards the control of the Tilt game play mechanic and allowed for the acceleration of the ball Sprite in the Math Wars game down the screen, and for the movement of the Number Line game’s helicopter Sprite from left to right. The Orientation sensor’s Pitch value reads 0 degrees when the device is level, increasing to 90 degrees as the device is tilted so its top is pointing down then decreasing to 0 degree as it gets turned over. Similarly, as the device is tilted so its bottom points down, pitch decreases to −90 degrees, then increases to 0 degrees as it gets turned all the way over (Android, 2013).
**Azimuth:** For this project, the Azimuth was integral to the design of the GPS control scheme and served the *Heading* parameter of the compass image. In this way, the compass image would point in the correct direction correct to the Azimuth degree. The *Orientation sensor’s Azimuth* value reads 0 degrees when the top of the device is pointing north, 90 degrees when it is pointing east, 180 degrees when it is pointing south, 270 degrees when it is pointing west, etc. (Android, 2013)

### 4.3.8 Location Sensor

The *Location Sensor* is a non-visible component within the App Inventor *Designer* providing location information, including longitude, latitude, altitude (if supported by the device), and address as long as location source is available. GPS and alternative methods such as cellular towers or Wi-Fi networks are considered to be location sources. For this project and the GPS control scheme implementation, the location sensor was forced into GPS mode and wasn’t given permission to access any other location source. This allows for the most accurate location reading possible and through rigorous testing, the *Location Sensor* was seem to be within 3 metres accuracy in optimal conditions (outdoors in an open area). (MIT, 2013)

### 4.4 Game Design Methodology

There was a concentrated focus on making the core game play simple and easy to understand, so that anyone would be able to pick up and play with little difficulty. It was the intention to make the games have interesting yet educational game play without them being unnecessarily complicated, and having minimal hand-holding and/or instructions.

The Math Wars game was built around the concept of space and travelling between planets and solar systems, while the Number Line Game was built around the idea of nature and of a helicopter attempting to find a patch of open area to land on, decided upon by a maths problem. The Screens within both games were built around each of these themes, with aesthetics of the buttons and backgrounds reflecting each game’s philosophy.
4.5 Math Wars Game

4.5.1 Design
For this game, a decision was made to force the game into portrait screen orientation. Seeing as the focus of the game was on the ball falling down the screen, it made logical sense to utilize more of the phone’s screen real estate, the extra length also giving additional time for the player to make a decision as to what side of the screen they want the ball to fall on.

The ball *Sprite* has a starting position at the top middle of the screen and returns to this point once it has collided with any area of the green wall containing the equality, located at the very bottom of the screen.

4.5.2 Game Mechanics
The goal of this game is to solve the problem within the falling ball *Sprite*, compare the result to the equality at the bottom of the Game Screen and then manoeuvre the ball to either left or right of the screen depending on the result.

The equality on the bottom of the screen is displayed and gives the player the opportunity to brush up on the basics of solving equalities. To the left of the number the greater than symbol is displayed, and on the right hand side, the less than or equal to symbol is displayed e.g. (< 4 <=).

The number in the middle is pseudo-randomly generated in accordance with the result from the maths problem inside the ball *Sprite*, which is also pseudo-randomly generated. Depending on the maths the player chose in the Pre-Game Screen, the values that are generated differ. The ball can be controlled by utilizing one of the two control schemes (Tilt/GPS).

4.5.3 Game Screen
Once the player starts a new game in the Main Menu and navigates through the Pre-Game Screen. Here, the player sees a 10 point number line at the bottom of the screen, a helicopter *Sprite* at the top of the screen, below the score, timer and maths problem.
If the selected control scheme is GPS, a small compass is displayed at the top right corner of the game area, which tracks the smartphone’s compass orientation through the Orientation Sensor. This gives the player an indication of what direction they are facing at any time, allowing them to re-orientate them back to North as needed.

4.6 Number Line Game

4.6.1 Design
For this game, a decision was made to force the game into landscape screen orientation. This was because it would allow for a much larger horizontal area than if the screen were to be in portrait like in the Math Wars game. The extra horizontal area meant that the number line on the bottom could take up a much larger proportion of the screen, and would therefore allow for more space between numbers, and allowing text to be more legible on smaller as well as large screens.

4.6.2 Game Mechanics
The goal of this game is to position the helicopter (top of the game area), and its corresponding red dot laser (bottom of the game area) on the correct number on the number line below. The player must solve the problems displayed at the very top of the screen that range from addition, to multiplication, to algebraic equations, and then move the helicopter/red dot ) to the result of this mathematical problem on the corresponding number on the number line by using one of the two control schemes provided (Tilt/GPS).

When the player manoeuvres the helicopter to their desired position, they must press the “Set Point” button, located directly underneath the number line. If the red dot lands on the correct resulting number (the result from the problem at the top of the game screen) the player will hear a correct “ding” sound, the correct answer will be displayed, and the problem will change from its initial yellow colour to green. This indicates to the player (both audibly and visually) that they are correct. On the other hand, if the player sets the red dot on an invalid
position and they are incorrect, a wrong “buzz” sound will play, and again the answer will display, however this time the colour would change to red. Again, this serves as an audio as well as visual indicator to the player, this time telling them they were incorrect in their position placement.

4.6.3 Game Screen

Once the player starts a new game in the Main Menu and navigates through the Pre-Game Screen they are greeted with the Game Screen. Here, the player sees a 10 point number line at the bottom of the screen, a helicopter Sprite at the top of the screen, below the score, timer and maths problem.

If the selected control scheme is GPS, a small compass is displayed in the bottom left corner of the game area, which tracks the smartphone’s compass orientation through the Orientation Sensor. This gives the player an indication of what direction they are facing at any time, allowing them to re-orientate them back to North as needed.

4.7 Shared Screens

Both the Math Wars and Number Line games feature very similar menus that provide the same basic functions to the player. The only difference being the user interface layout and aesthetics such as the buttons and backgrounds. The Main Menu, Pre-Game Screen, Game over Screen, and High Score Screen all follow the same design guidelines.

4.7.1 Main Menu Screen

The Main Menu screen serves as the central hub for the game(s) and is accessible from any of the other screens. Here, the player has the option to begin a new game, resume an existing game (if available), and transition to the High Score Screen where they may view their current high score. The Resume button is only visible when the player has started a new game at a previous point in time, and
restores their progress from that instance by way of the TinyDB that stores the Sprite position, score, timer value, and other significant variables.

![Number Line Game](image1.png)  ![Math Wars](image2.png)

Figure 11 – NL Game Screen  Figure 12 – MW Game Screen

### 4.7.2 Pre-Game Screen

The Pre-Game Screen appears after the New Game button is pressed in the Main Menu. This screen allows the player to select the scope of mathematics they want to tackle in the game proper. The buttons are designed as toggles that give the player visual recognition when they press them. After the player has made a selection, they must then choose a control scheme before being able to start the game. Since only one control scheme can be active at any one time, the “GPS” and “Tilt” buttons are mutually exclusive in both a visual and logical sense. The Notifier component is programmed to display a set of prompts for the player indicating if they haven’t chosen a control scheme or if no math is selected. They are also prompted to confirm their selection if the above criteria are met.
The player also has the choice to return to the *Main Menu* by way of the *Main Menu* button located on the top right of the screen.

Figure 14 – Pre-Game Screen

Figure 13 – Pre-Game Screen

### 4.7.3 Game Over Screen

The *Game Over Screen* is displayed within the *Game Screen*. When the game timer reaches 0, the game resets by moving the *Sprites* back into their original position and setting the timer and score values back to their defaults. The screen is cleared, and the player’s final score is displayed along with “Game Over” text. The player is then given the opportunity to start a new game by pressing the *Go Again* or *Play Again* button, or may return to the *Main Menu* as before.

If the selected control scheme is GPS, the compass component then re-appears and the player must orientate themselves to North once more.

Figure 15 - MW Game Over Screen

Figure 16 - MW Game Over Screen
4.7.4 High Score Screen

This screen simply displays the highest score from any games played on that device since the app was installed. The high score is extracted from the TinyDB component. When a game is over, there is a method within the Game Screen that checks whether the final score for that game is greater or less than the current high score.

4.8 Maths Problems on Screen

Within both games, the player is able to pick what maths they want the game to generate as they play; the only difference is the way this is displayed. In the Math Wars game the player has to keep track of two numerical functions on the screen. The maths problem is generated and printed inside the falling ball Sprite, and a subsequent equality is displayed at the bottom of the screen on top of a wall Sprite object. Conversely in the Number Line game, only the problem needs to be displayed at the top of the screen, with the numbers underneath the number line generated and printed also. The helicopter and the red dot displayed on the number line itself are the position indicators for the player.

4.9 Points/Time System

The points system functions in largely the same fashion for both games. If the player answers a problem correctly, they are awarded with 5 points, and this is added to the total game score at the top left of the game screen and the game play resumes immediately (Math Wars) or waits for the player to go for another round.
(Number Line). On the other hand if the player answers incorrectly, there is no change to the game score and the game continues as before. The game keeps track of each consecutive correct answer, with the player being rewarded if they continue to answer correctly in succession whereby the player can receive bonuses to their points score via a score multiplier. The score multiplier increases up to a total of 5 times, meaning the player can receive up to a total of 25 points per correct answer. The game’s score multiplier is reset whenever the player answers a problem incorrectly or if the timer runs out, in which case the game ends.

Both games are both guided by a real-time game timer that decreases by 1 every second. This allows the player a set amount of time to play the game once a new game is started. The timer for both games is initially set at 60 seconds which gives the player enough time to get a handle of the game mechanics and controls if they are a first time user, while at the same time giving return players a challenging time constraint to test their abilities. The games being on a mobile form factor also contributed to this decision. Considering successful games such as Angry Birds which has been downloaded over 1 billion times as of May 2012 (Hartley, 2013) offers bite-sized chunks of game play that last mere seconds and align themselves favourably to the portable nature of a smartphone. A quick 60 second fix of mathematics may not be as daunting or as boring to children be it in a school or out of school environment.

4.10 Control Schemes

The Math Wars Game and the Number Line Game were designed around two control schemes – Tilt and GPS. Below, the two are outlined in a more detail.

The Tilt control scheme (utilizing the orientation sensor) was designed to be the simpler and faster paced method of the two for controlling the game Sprites. The Tilt control scheme places the player in a more reactionary position as opposed to the GPS control scheme where the player has a little more time to react to the game play.
### 4.10.1 Tilt (Orientation Sensor)

The Tilt mechanic allows the player to control the movement of the game *Sprites* accurately as the phone’s orientation sensor is able to measure the smallest of movements by way of the sensor’s pitch and roll readings. Hence, it also gives the opportunity for more precise control as compared to the GPS movement control scheme. It therefore makes it quite easy for a novice player to pick the game up and play without much difficulty due to the tilt control mechanic having a relatively low learning curve; the player just has to tilt the device in a specific direction.

The Tilt control functions by way of the pitch and roll parameters located within the orientation sensor component. The orientation sensor tracks the magnitude of movement produced by the player and acts accordingly. For the Math Wars game, the movement mechanic is a little more complicated because the ball automatically moves from the top of the game Canvas down to the bottom, where it hits a wall and re-appears at the top again. The player has the ability to let the ball glide down the screen at its normal pace as long as the phone is in a flat position, with the bottom facing the ground/floor. The player can then tilt the phone left and right of that position, and the ball will follow their movements. The player can also accelerate the movement, but only to a certain extent. This game mechanic was added in case the player is quick to solve the problem and they are confident in their choice to move the ball to either side of the screen. They can then tilt the phone downwards so that the bottom of the phone point to ground at an angle if they wish to speed up the fall speed of the ball *Sprite*.

However, as previously stated this is controlled by the player and there is a limit placed on the highest potential speed the ball can fall.

Therefore, players who find the problems easy to solve or are just more knowledgeable on the maths they select can cut through the sections they find easy and move onto the next problem. Because there is a strict time limit, the player cannot easily exploit this “increased” movement speed if the player just decides to guess the answers. The speed/tilt limit is placed in order to prevent this from happening.
4.10.2 GPS (Location Sensor)

For the GPS control scheme to function as needed and for it allow for on-screen Sprite movement, the games needed to be reworked in terms of the logical design, game mechanics combined with a few additions to the Game Screen that had to be made. The biggest addition was that of a small and a large Compass that appears for the player right before the game initializes, soon as the player selects their mathematics and select “GPS” as the control scheme. The Compass elements are displayed along with the usual game components, and these play a large part in the setting up of the GPS control scheme.

For my implementation of GPS movement tracking, the player must orientate their phone to face North on the compass. This is done by navigating the “N” point on the Compass so that it faces the top of the phone screen (be it landscape or portrait orientation). When the player faces north and presses the “set north” button in the game, they align themselves with the latitude (horizontal world lines) and the Prime Meridian. When this occurs, the timer is started and the game Sprites are unfrozen and able to move around freely. A method constructed within the Game Screen is able to track a user’s movements left to right by moving from left and right of this set position. An increase in longitude indicates that the player has moved right, while a decrease in longitude indicates a move left.

By moving to the left and right of this set point, the player is actually navigating along the lines of longitude along the earth’s axis. This is accomplished by tracking changes in the longitude of the phone’s GPS co-ordinates and then moving a certain number of pixels over a period of time to simulate smooth movement. Since the location sensor is set to track changes in the GPS co-ordinates over an interval of 3 metres (set in the Blocks Editor), it requires about 4-5 steps for the location sensor to identify movement before it can track a change in location and move the Sprite accordingly. Through testing, setting this parameter to anything lower than 3 metres resulted in inaccurate readings as the location sensor’s Accuracy parameter itself being erratic at times with the Accuracy changing from being 3 metres accurate one second, to 8 metres accurate the next.
It is also necessary to note that the game will only function as intended if the player finds an open area to work with. An open green area is recommended, but the games have been testing in a more congested environment with surrounding buildings and the movement was still tracked within reasonable accuracy.

4.10.3 GPS Control Scheme - Game play Implications

In the Math Wars game if the player is playing with the GPS control scheme selected, the Sprite is purposefully designed to move at a slower pace. This design choice was necessary as the ball was found to be moving too quickly along the game canvas and the player simply did not have enough time to respond to and solve the problems given to them.

Conversely, in the Number Line Game since the helicopter and its red dot tracker can only be controller directly by the player, such a change was not necessary. Instead, the helicopter is designed to move to the next number (rung) along the number line if the player moves in any direction. The player has the ability to stop the game Sprites whenever they feel they have reached the correct position.

4.11 Technical Challenges

The inability to draw text on top of image Sprites was a major technical challenge; something that can easily be implemented in a high level language without much difficulty. This, as stated earlier when discussing App Inventor in general is one of the inherent limitations of the environment. It created a problem from a design standpoint as being able to display text on the ball and wall Sprites was an integral part of the Math Wars game. Without being able to draw text inside the descending ball Sprite, the game’s mechanics would not have turned as intended.

Movement through GPS as a control scheme was quite difficult to implement and there was a significant amount of time devoted to this aspect of the project. In the early design stage, questions such as these had to be answered before development could start: Will the player have complete control over the game Sprites? How will the game handle changes to the GPS co-ordinates? And, how will this all be displayed on screen?
4.12 Solutions to Challenges

To overcome the challenge of displaying text on top of Sprite images for both the ball Sprite and wall Sprites, hidden Canvas Components has to be utilised. This hidden Canvas component would be used to store text values that were not visible to the player by drawing the text, then saving it as an image file to the device’s usable storage memory. Once this was done, any ImageSprite Component was able to display the saved values and was then programmed to follow the ball Sprites X and Y position and overlaying the newly created ImageSprite over the ball and wall Sprites. However, this was certainly not an ideal solution despite the stored images being insignificant in size (~400 bytes).

For the GPS control scheme implementation, it was decided to limit the player’s control to just the Y-axis within the game area for both games. This made logical sense for the Number Line game as the player would only be required to move along the number line itself and nothing else. For the Math Wars game where there happens to be a higher level of complexity, the same movement limitation was put in place because total Sprite movement control would require a very large area for movement as well as a re-thinking of the GPS movement implementation. And so, the player was confined to being able to move the player-controlled game Sprites from left to right only along the X-axis. There needed to be a way to track a player’s movement in an open area, and thankfully App Inventor provided developers with an in-built method within the location sensor component (LocationSensorMoved) which tracks and detects whether there has been any change in the latitude and longitude co-ordinates that activates when the player has moved and the location sensor tracks this movement over a certain distance (Distance Interval parameter).

4.13 Discovered App Inventor Limitations

Throughout development, it was imperative to note any inherent limitations or “ceilings” that App Inventor may have as a programming construction kit. Such limitations have value to any App Inventor existing and/or prospective developers. Below are some of the issues or limitations that were discovered and are not listed on the MIT website, as well as the issues that are known.
4.13.1 User Interface Design Limitations

There is some difficulty in building a fully featured user interface with the App Inventor Designer and arranging those assets in a way one would desire. For example, a user can place down Components such as buttons on the screen but they are immediately forced into a default area without the explicit ability to move these components to a user-defined location. While there are ways to overcome such a problem, it relies on placing horizontal and vertical Screen Arrangements in order to align components. The use of Label components is also necessary for careful positioning. These allow the developer to alter the positions of other objects by providing the necessary width or height difference required as empty space. This makes positioning of components troublesome and involves a significant amount of unnecessary trial and error.

4.13.2 Android User Interface Guidelines

While App Inventor does support the latest version of Android (Android, 2013), the components it supports tend to visually appear based on an older version of Android at its core. Some in-built functions like the Notifier component that allows for the displaying of an assortment of on-screen notifications and pop-up dialogue boxes, as well as basic components such as the text box, check box and button, do not follow the official android development guidelines set out by Google (Android Developers, 2013). Therefore, the applications that are currently possible within App Inventor will as of this moment do not to look as clean or professional as Android applications created through the Android SDK, an appropriate IDE and Java.

4.13.3 Components required to be run locally

Some features/components require that the user compile and install the application directly onto an Android device in order to test certain features or components. The latest version of App Inventor still does not support the switching of screens through the Blocks Editor with a connected Android phone or through emulator. This also implies that the internal database component (TinyDB) is affected, including other components such as the Location Sensor for tracking GPS co-ordinates. For more complex applications that require multiple screens, the TinyDB component, or the use of the Location Sensor the
developer must have direct access to an Android smartphone and the application has to be complied and run locally on the device.

### 4.13.4 Blocks Editor Issues

As the application gets progressively more complex and more blocks created and placed within the confines of the Blocks Editor, the editor itself becomes somewhat laggy and less responsive than it is when first starting new application or early on in development when there are few block to manage. It can take longer to conduct simple actions such as dragging and dropping blocks, inserting blocks into command blocks, or generating blocks through typing.

The blocks within the Blocks Editor are not easily replicated. The copy function, while useful in some circumstances is a little unintuitive and it is oftentimes easier to type in blocks by left clicking and finding the desired block. Another downside is that the Blocks Editor does not allow for the copying of blocks or block snippets between screens or between projects. This makes it troublesome to copy code from one place to another. While blocks are more visual, they take up a lot of screen space making it difficult to arrange and sort through blocks when programs get increasingly more complex. Textual programming in this regard, is easier to manage.

### 4.13.5 MIT App Inventor Known Issues

It is necessary to note that App Inventor is still a product in its beta stages, and therefore has issues that are known to MIT. This implies that App Inventor is still under development, with changes and fixes being made on a regular basis. Some issues that are known and displayed on the App Inventor website (MIT App Inventor, 2013) are:

The Blocks Editor is unable to load large projects. If an attempt is made to transfer over a large project with a large number of image files for example, the error "Unable to load project" would be displayed.

The Blocks Editor can become confused about which screen is currently being working on if the developer tries to manipulate any blocks in between switching screens or projects, and this can result in errors or even the loss of blocks. It is
recommended to wait until the Blocks Editor has finished loading before resuming work with the blocks again. This issue occurred several times throughout the course of this project and resulted in the loss of blocks and work. Thankfully, manual backups were made and it was still possible to regain the lost data without too much difficulty. The backing up of projects is therefore highly recommended.

Additionally, there can be unstable project loading: There have been reports of projects loading incorrectly, or with blocks missing. Again, it is recommended to be diligent about keeping online and local backups, especially for large and important projects.

Failures to package projects are also quite common. A 6% failure rate in attempts to package projects has been identified by MIT and many of these appear intermittent. If a project fails to package, it is suggested to try again several times to package the project again. Another source of failure may result from case collisions in the names of media files (common names), and errors may be caused by this also.

4.14 Design Summary

The two games were developed over the course of 12 weeks throughout the college year. Functionality-wise, the two games matched the initial design goals of developing maths-based educational games using a selection of control schemes. However, due to time constraints some bugs yet remain and the user interface between the two games could be differentiated in some way other than aesthetics, as currently they look too similar.

The GPS control scheme and its implementation was by far the most time-consuming aspect of the project and only saw a finished working version in the mid-late stages of development. The inability to overlay text on Sprites was also an on-going concern throughout the project as it was an integral part of the game play mechanics for the Math Wars game. This problem was only solved towards the end of the coding stage and meant the reworking of the way maths problems and text were displayed on the screen.
Some game play additions were thought of, but were left due to priority being placed on the more important functions of the games, primarily the GPS control scheme implementation. The games lack options for controlling the level of difficulty, be it sprite movement speed (Math Wars) or the scope and difficulty of maths problems available (both games). There is room for future additions, and these will be discussed further in the conclusion.
Chapter 5 - Evaluation

5.1 Evaluation Methodology

The game applications were evaluated with the aim of assessing their overall quality as well as their effectiveness as educational tool. Having this in mind, feedback was sought from two groups of students participating in the Bridge 21 Scheme. Bridge 21 is a joint venture of Trinity College Dublin and SUAS Educational Development that offers a new model of technology-mediated learning with the core goal of enabling a new space for learning in the 21st century (Bridge21, 2013).

The first group was the larger of the two, consisting of twenty two participants, and the second was the smaller group of seven. Testing had to be conducted on two separate occasions due to the uncompromising weather on the initial test day. Therefore, two individual sections outlining the results of both testing occasions will be described below.

The games were designed to provide students with an engaging and hands-on way of interacting with mathematical topics. The data collected from both sets of sources was of a qualitative form and was in the form of a SUS test and a series of custom-made questions utilizing the likert scale as a basis for measurement. The results gathered from the testing would indicate whether the pupils thought the apps were any fun and if these games had any perceived practical implications for their mathematical studies in school. Recommendations and suggestions for further improvements were also part of the evaluation process and participants were encouraged to comment or leave any feedback at the end of the questionnaire process.

5.2 Testing on Students - Overview

5.2.1 SUS Test and Questionnaire

A System Usability Scale (SUS) test was conducted along with a selected of personally made likert scale based questions that directly related to the two
games (see appendix for questions and results). The SUS is a simple, ten-item scale giving a global view of subjective assessments of usability (Brooke, 1995). SUS scores have a range of 0 to 100. The average SUS score from all 500 studies is a 68. A SUS score above a 68 would be considered above average and anything below 68 is below average (Sauro, 2011).

### 5.2.2 Sample Selection

Evaluation of the games were in the form of testing on multiple android devices, making sure that both Math Wars and the Number Line Game were universally built and functioned in the same way on any device. This was then followed by hands on testing by a group of testers in Bridge 21 by a collection of secondary school students aged 14-16. Bridge 21 has ethics approval for carrying out evaluations with its participating students, and the Program Co-ordinator (Kevin Sullivan) ensured all necessary consent forms were provided and signed.

### 5.3 Test Day 1

#### 5.3.1 Tilt control scheme

For the first test day, 22 students were recorded as participants. The participants were handed out test devices to try both the Math Wars and Number Line games. However, due to time constraints some were only had the opportunity to test only one of the two games. All students had the opportunity to test the Tilt control scheme within the games they tested. Students were given roughly 5 minutes per game each, and were then asked to fill out a SUS test and questionnaire as described above.

#### 5.3.2 Test Day 1 Feedback - SUS Results

The SUS test results were collated and measured giving a resulting SUS score of 70.34 (Appendix 1). This score is above average for a SUS test, and indicates an overall positive attitude of the participants in relation to the general usability of the two games. Most participants had the opportunity to test both games, and so this score is reflective of the usability of both the Math Wars and Number Line games.
5.3.3 Test Day 1 Feedback - Questionnaire Results

With the results only being based off of one control scheme, the results gathered from Test Day 1 were incomplete (Appendix 3). While they did provide an idea of what the participating students though of the games in a general sense, with the students scoring mostly high on the likert scale for enjoyment received, and similarly on whether they thought these games would help them with their school studies.

Overall the results indicated that the games were still of interest to students at the ages of 14-16. However, the students did not think the games were very difficult, the results skewed quite low on the likert scale. It is important to note that most of the students seemed to choose easier maths such as addition and subtraction and many did not select multiple types of problems, instead staying with just one.

5.3.4 Comments

(1) Asked to improve the visual effects
(2) Points system was at times inconsistent.
(3) Helicopter game sprite was tricky to set down in the desired position, implying that the orientation sensor wasn’t accurate.
(4) Add a “how to control” menu or tutorial as some participants did not know how to control both games without being shown.

5.4 Test Day 2

5.4.1 GPS/Tilt control scheme

For the second test day, a group of 7 students were gathered to test both games and control schemes. Participants were given roughly 5-8 minutes each to play both the Math Wars and Number Line Games, so that they became familiar with the two games and the Tilt control scheme. Then, they were taken outside to an open area in order to provide the participants enough room to move around. This was necessary to achieve optimal GPS co-ordinate accuracy when using the GPS control scheme. Once the phones had locked their GPS co-ordinates, the testers were again given roughly 5-8 minutes to play the games. As before, they were
asked to fill out a SUS test and questionnaire and were encouraged to leave more detailed feedback and comments

5.4.2 Test Day 2 Feedback - SUS Results

The SUS test results were collated and measured giving a resulting SUS score of 70 (Appendix 2). This is comparable to the Test Day 1, and indicates there is a general consensus regarding the usability of the applications. As before, this score is above average for a SUS test, and indicates an overall positive attitude of the participants in relation to the general usability of the two games.

5.4.3 Test Day 2 Feedback - Questionnaire Results

This time all of the participants had a chance to test out both control schemes (GPS/Tilt) and so the questionnaire conducted on Test Day 2 produced a broader and more transparent set of results (Appendix 4).

Most participants were impressed by the GPS control scheme and thought it would help them with their maths studies in school. The general outlook was that these games were indeed fun, whether in a school or out of school environment.

However, the students did not think the games were very difficult, the results skewed quite low on the likert scale. It is important to note that most of the students seemed to choose easier maths such as addition and subtraction and many did not select multiple types of problems, instead staying with just one.

5.4.4 Comments

(1) The GPS control scheme seemed difficult to use without any prior guidance.
(2) Some participants were very satisfied with GPS control and found it interesting and useful as a learning tool.
(3) One participant noted that while the GPS was interesting, they preferred to be able to play wherever they are, which the Tilt control scheme allows.
(4) Some bugs were discovered such as text printing over other text.
(5) An exploit was discovered in the Number Line game that allows the player to spam the go again and set point buttons in one position until the player receives a correct answer.
5.5 Accessing Feedback

The feedback provided by the students was very helpful and highlighted some apparent issues and improvements that can be made to both games. Some things can be easily changed, added or fixed, while others are outside the scope of App Inventor’s or of a smartphone’s sensor abilities.

Improving the visual effects may be done to an extent, but this is heavily limited within App Inventor as the developer is confined to using the tools that are provided, or by importing their own images to use as buttons, background, or Sprites. 3D animation is not possible.

The games were originally designed on a powerful dual-core Android smartphone and suffered no issues with game jerkiness or lag. For the Sony Xperia U phones that were being used as test devices, the orientation sensor was less forgiving and the Tilt movement remained less responsive for the first few play attempts of the games. However, the accuracy increased and the movement became smoother once the devices adjusted to the movement. A similar issue was observed with the Math Wars game, but again, it became less evident after a few play attempts. As the logic and components were built within App Inventor, there is little to no control over issues such as this occurring. Therefore, there is no foreseeable way to fix this.

A control menu with instructions on how to use both control schemes was suggested by one of the participants. This is definitely something that would make the games a more accessible and would give the player a better concept of what the games are about, and how the game mechanics work. Again, this can be added in the future without much difficulty.

An exploit was discovered within the Number Line game that allowed the player to simply repeatedly tap the Set Point and Go Again buttons while hovering over one spot on the number line. Eventually, they would get a correct answer and receive points for doing so. As the game does not penalize the player for being wrong, it is very easy for them to not follow the logic of trying to solve the
problem at the top of the screen and use this exploit to attain an easy high score. To avoid this, a penalty could be presented to the player for being incorrect or perhaps a timer that limits the use of the Go Again button’s function.

5.6 Evaluation Summary

Overall, the testing and feedback was generally positive and did not reflect badly on the two games developed. Most criticisms can be worked on and overcome within the constraints of App Inventor. The last chapter, Chapter 6 will conclude this report and outline any potential future work.
Chapter 6 - Conclusion

The previous chapter explained how the games were evaluated and outlined the results from that evaluation. This chapter draws some conclusions from those results and highlights some criticisms of the project. The future work planned for the project is then detailed.

6.1 Results

6.1.1 Were Useful Educational Games Developed?

Based on the testing results and feedback, the participants were satisfied with both Math Wars and the Number Line Game on both usability and game play standpoints. The project’s objective was to evaluate App Inventor as a powerful low-floor, high ceiling programming tool, the development of maths-based games essentially being a stepping stone and framework in the exploration of this goal.

However, the combination of education and m-learning are valuable in a sense that they provide new experiences for children that have previously been undervalued and unused within many education systems. Therefore, any advances made in this area are of great interest and benefit to education.

6.1.2 How effective is App Inventor as a low-floor, high ceiling tool?

App Inventor can be an effective low-floor, high-ceiling programming tool from the experience gathered from this project. However, it is important to note that each comparable tool outlined in Chapter 3 serves a different purpose, and can therefore only be compared subjectively.
Having a programming background in Java prior to this project, it can be said that it was easier to understand and get to grips with the concepts and the core components within App Inventor. If someone with no prior experience were to attempt to develop within its environment, the challenges, outcomes and development timeframe could be entirely different. To further the research of this project, observing less experienced users attempt to development similar Android applications within the App Inventor would prove useful.

### 6.2 Criticisms

#### 6.2.1 Testing Sample

The testing sample consisted of 14-16 year old male and female students. This represents a slightly older age group than for whom the applications were designed for and were therefore not the most optimal of test subjects. However, in terms of testing the game play mechanics and usability of the games, this particular age group was suited to the task and managed to provide appropriate feedback through testing.

#### 6.2.2 Mathematics Difficulty

Across the two games, a selection of mathematics was chosen to be implemented. The scope of mathematics included addition, subtraction, division, multiplication, equations, fractions and integers. This covers the basics of maths and players can select as few or as many of these as they wish. The easier maths allows for primary school children to play too from as early an age as 9 or 10, while fractions and equations give older children 11+ in an opportunity to test their abilities also. The games do not have an explicit difficulty level, however the player easier/more difficult maths problems may can be added in a future expansion.

#### 6.2.3 Scope of Mathematics

The mathematical topics were chosen as a suitable instrument for learning due to the subject’s importance in modern day school curriculums. However, this does not mean that similar games with alternative or additional mathematical topics
would not be viable, or that topics outside of the subject of mathematics would not translate into useful and educational games.

6.2.4 Scope of Development

One clear limitation of the games developed as part of this project was the fact that the games themselves weren’t developed to a fully finished state, and are still not feature-complete. While it was felt that they were developed sufficiently to investigate the research question, the games were and still are relatively underdeveloped. The games are simplistic in their execution and many more enhancements can potentially be made to increase their usefulness. The project needed to be completed within strict time constraints and this was the main obstacle to further development. It took several weeks to visualize the basic underpinnings and initial designs for the games, and the GPS movement controls took up a large proportion of the development time as the implementation saw several redesigns and required rethinking and iterating upon to arrive with a functioning solution.

6.3 Future Work/Improvements

6.3.1 Expansion of Applications

The number line game can always be expanded to contain a broader range of mathematical problems. Fractions were not implemented in the Number Line Game due to time constraints and could have added a further level of complexity. Other, more complex mathematical problems can also be added; more algebra, for example. The Math Wars game may also undergo the same treatment. However, the faster-paced nature of this game compared to the number line game still means that problems cannot be as difficult as what is potentially possible in the Number Line game, and so they have to be solvable within a very short time span. Therefore, extensions to the scope of mathematics can be made, but should be well thought out and tested beforehand.

The number line concept in the Number Line game can be expanded to co-ordinate geometry whereby the player must move around and think within the bounds of both the X and Y axis. However this in itself would bring a host of
challenges to overcome in terms of GPS control movement and game play mechanics, as well as requiring alterations to be made for the aesthetic look and feel of the game.

6.3.2 How to Play Screen

One of the suggestions in the feedback asked for the adding of an instructions or “how to play” screen, detailing to the player how to play and control the games through either control scheme. This would aid in the general usability of the games, and allow for more people to play them without any prior knowledge or third party aid.

6.3.3 Combining the two applications

As the two applications were not developed concurrently (the Math Wars game first, number line second), they were developed as two separate and singular entities. While some block-code was re-used in the Number Line game, the games were not thought of as a suite of applications. However, the two games can essentially be combined into one application in the future, with the intention of adding other interesting and different games to the package. This could potentially enhance the experience and allow for seamless transition between the two games.

6.3.4 Suggestions/Final Thoughts

Overall, there are certainly potential improvements to be made to the applications and programming tools such as App Inventor. A suggestion for MIT and App Inventor would be to develop a desktop application that allows for the interaction of the Designer and the Blocks Editor, and perhaps even eliminating the need of a Google account. This could perhaps lower the floor of App Inventor even further.
Bibliography


Alice, 2013. *An Educational Software that teaches students computer programming in a 3D environment*. [online] Available at: <http://www.alice.org/index.php>


Appendix

Appendix 1

Page 1 - System Usability Scale

<table>
<thead>
<tr>
<th>Name:</th>
<th>Gender:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Strongly Disagree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this app frequently</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>2. I found the app unnecessarily complex</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>3. I thought the app was easy to use</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>4. I think that I would need the support of a technical person to be able to use this app</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>5. I found the various functions in this app were well integrated</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this app</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this app very quickly</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>8. I found the app very cumbersome to use</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>9. I felt very confident using the app</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with this app</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
</tbody>
</table>

Page 2 – Post-game Questionnaire

What maths game(s) did you play? Math Wars / Number Line

1. I enjoyed playing the game(s) 2. These games would help with my maths studies at school
3. I could see myself playing these games in my free time 4. I found the maths problems to be sufficiently difficult
5. I preferred playing the game(s) with the GPS control scheme
6. I found the GPS control scheme to be an interesting new way to play the game(s)

Please leave any additional comments below:

________________________________________________

________________________________________________

________________________________________________
Appendix 2 – SUS Test Results Day 1

34
34
32
31
34
35
22
18
29
32
26
29
31
25
20
27
30
12
19
36
32
31

28.14 * 2.5 = 70.34 SUS Score

Appendix 3 – SUS Test Results Day 2

34
11
33
28
29
35
26

28 X 2.5 = 70 SUS Score
## Appendix 3 – Post-game Questionnaire Day 1

<table>
<thead>
<tr>
<th>Game(s) Played</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Number Line</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Both</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Math Wars</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Number Line</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

## Appendix 4 – Post-Game Questionnaire Day 2

<table>
<thead>
<tr>
<th>Game(s) Played</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Both</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Math Wars</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Both</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Both</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Both</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Number Line</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>