The Development and Evaluation of an Electronic Serious Game Aimed at the Education of Core Programming Skills

by

Leon van Niekerk

MA (Socio-Informatics)

Thesis presented in fulfilment of the requirements for the degree of Master of Arts in Socio-Informatics in the Faculty of Arts and Social Science at Stellenbosch University

Department of Information Science
Stellenbosch University
Private Bag X1, Matieland 7602, South Africa

Supervisors:

Prof. Bruce W. Watson Prof. G-J van Rooyen

December 2016

Declaration

By submitting this report electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2016

Acknowledgements

Acknowledgements need to be given to the following people and organisations for their assistance with the production of this thesis:

- Prof Bruce Watson for his guidance and supervision.
- Prof. Gert-Jan van Rooyen for his supervision and guidance.
- Richard Barnett for his help in data gathering.
- Maryke de Wet for her help with language editing.
- Naspers and the MIH Media Lab for providing an amazing working environment.
- The Department of Statistics and Actuarial Science at Stellenbosch University for their help with data analysis.

Abstract

English

The integration of information technology with everyday life has increased the demand for the number of programmers and computer scientists, yet the number of students moving into these fields professionally has not kept up with this demand.

Education, and fostering interest is one potential way to increase the number of students moving into these fields. While good teachers and schools can develop this interest in students, this research explores the use of an educational serious game to both teach students the fundamentals of programming, while also increasing their interest in the field.

Serious games are digital games with a primary purpose other than entertainment. In the case of this research, the purpose is education.

A prototype serious game was developed to teach students the concepts and processes involved in programming and algorithmic development, rather than the writing of programming code. Abstract symbols represent blocks of conceptual code, which can be manipulated by the player in order to "program" solutions for predefined problems.

In addition, the research called for testing the prototype. For this purpose, introductory programming students at the University of Stellenbosch were approached as test subjects. These students were asked firstly to complete a language-agnostic programming aptitude questionnaire, also developed as part of the research, at the start and end of their semester; and secondly, a subset was asked to play the game during the semester.

Several metrics were gathered from these tests, namely, their university marks for the course, the results of the language-agnostic aptitude test, the previous programming and mathematics experience of the students, and an opinion ques-

tionnaire from the subset of students who played the game.

While student fallout throughout the course was expected, the small class size and voluntary nature of their involvement in the study led to an unexpectedly low number of usable data points. However, it was possible to obtain the course marks from the students without their involvement. Thus, the test results were used in conjunction with the valid university course marks to establish a conclusion.

Students who played the prototype scored significantly better in the quantitative tests than those who did not. This in combination with the results of other earlier studies indicate that games can be used as tools for the enhancement of the learning process.

Afrikaans

Die integrasie van informasietegnologie met die alledaagse lewe het gelei tot 'n toename in die aanvraag vir programmeerders en rekenaarwetenskaplikes. Terselfdertyd het die aantal studente wat professioneel in hierdie velde inbeweeg nie bygebly met hierdie aanvraag nie.

Opleiding, asook die aanmoediging van belangstelling in hierdie velde is maniere om die aantal studente te vermeerder. Goeie onderwysers en skole kan ook moontlik hierdie belangstelling kweek.Hierdie navorsing ondersoek egter die gebruik van 'n opvoedkundige, ernstige speletjie om die kernkonsepte van programmering oor te dra, asook belangstelling onder studente te kweek.

Ernstige speletjies is digitale speletjies met 'n ander primêre doel as vermaak. Vir hierdie navorsing was daardie doel opvoedkunde.

'n Prototipe van die ernstige speletjie was ontwikkel om studente te leer van die konsepte en prosesse betrokke by programmering en algoritme-ontwikkeling, eerder as die skryf van programmeringskode. Abstrakte simbole stel blokke konseptuele kode voor, wat kan beheer word deur die speler om oplossings tot vooraf bepaalde probleme te "programmeer".

Die navorsing het ook vereis dat die prototipe getoets word. Vir hierdie doel, was inleidende programmering studente aan die Universiteit van Stellenbosch benader as respondente. Hierdie studente was gevra om 'n taal-agnostiese programmeringsaanlegvraelys te voltooi aan die begin en einde van die semester.

Hierdie vraelys was ook ontwikkel as deel van die navorsing. 'n Onderafdeling van die groep studente was ook gevra om gedurende die semester die speletjie te speel.

Verskeie maatstawwe was versamel van hierdie toetse, naamlik die studente se punte vir die kursus, die resultate van die taal-agnosties aanlegtoets, hul vorige programmering en wiskunde ervaring, en 'n meningsvraelys uit die onderafdeling van studente wat die speletjie gespeel het.

Terwyl dit verwag was dat studente sou uit val gedurende die kursus, het die klein klasgrootte en vrywillige aard van hul betrokkenheid by die studie gelei tot 'n onverwags lae aantal bruikbare datapunte. Die studente se klaspunte was nietemin beskikbaar sonder hul vrywillige inset. Die toetsuitslae is dus gebruik, saam met die geldige klaspunte, om 'n gevolgtrekking te vestig.

Studente wat die prototipe gespeel het, het op 'n statisties betekenisvolle vlak beter punte behaal in die kwantitatiewe toetse as dié wat dit nie gespeel het nie. In kombinasie met die resultate van vorige studies dui hierdie resultaat aan dat speletjies gebruik kan word as instrumente vir die verbetering van die leerproses.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Gamification, or gameful design, is the practice of incorporating elements of game design into systems that do not require them, in order to increase the level of user engagement with the systems. To some extent, this practice can theoretically be applied to any system, not just digital ones. Serious gaming, on the other hand, is the use of a game for any primary purpose other than entertainment. These purposes include, but are not limited to, advertising, education, recruitment and information gathering. Both gamification and serious games make use of games: serious gaming as a whole, whereas gamification uses individual elements of game design.

The research presented here demonstrates that serious gaming in particular, shows promise in teaching students abstract skills, such as computer programming. It is hoped that the advantage of user engagement offered by these approaches will prove useful in promoting an interest in computer programming as a field of further study.

This research then presents three sections. Firstly, an overview of the relevant literature and past attempts at the problem of educating through game-play. Secondly it describes in detail an experiment whereby students enrolled in an introductory programming course were exposed to a serious game prototype containing the same concepts in addition to their studies. Lastly it presents the quantified results of said experiment as well as conclusions that can be drawn from it.

It is concluded that the results gathered show a correlation between exposure to a programming game and results in university course marks. However, due to the inherent exploratory nature of the research and the unintended small size of the testing group, further research is recommended to be conclusive.

1.1 Problem Statement

Trends over the past decade indicate that the number of students which are moving into the computer science field, or fields related to it are decreasing. Surveys administered at the University of California, Los Angeles saw a decline in Computer Science majors amongst freshman of 70% between 2000 and 2005 (Crenshaw et al., 2008). Similarly, the interest shown by and results of high school learners' science courses have also declined in South Africa (Muwanga-Zake, 2003).

Even as this decline in the number of prospective computer scientists is happening, dependence on computer- and internet-based technologies continues to grow. More and more, existing technologies are linked together and computer chips are finding their way into a greater variety of objects. There is no indication that this trend is slowing down (Conti, 2006; Smith, 2011).

This presents a serious discrepancy between the number of new professionals being produced and the number of software professionals required in the world at large. This discrepancy needs to be addressed before the cost of developing new software becomes too high and the number of knowledgeable individuals too low.

Shrinking this discrepancy represents a serious undertaking. It deals not only with increasing the number of learners and students who have the required technical knowledge but also with fostering a lasting interest in these fields amongst school and university students so that they eventually enter these fields professionally.

Stimulating the technical knowledge as well as the interest of learners and students is an ongoing process. It follows that one of the most crucial periods to foster this kind of interest is at the beginning of a learner or student's exposure to the field. While technical expertise can be improved upon throughout the learning process, the student must be available, and keen enough, to be taught.

The research presented here focuses on electronic games as a medium of education. Specifically, it deals with the creation of a serious game, that is to say, a game with a primary purpose not of entertainment but of informing on the basics of computer programming.

1.2 Research Goals

This research makes use of an electronic serious game as an aid for computer science learning.

1.2.1 Thesis Statement

The goal of this research is to answer the following questions:

1. Can a digital serious game be used to **enhance** an introductory programming course to increase a student's understanding of relevant concepts?
2. Can a digital serious game be used to **replace** an introductory programming course to establish a student's understanding of relevant concepts?

As such, the second research question is inherently a stronger version of the first question.

1.2.2 Hypotheses

Given the research questions in the previous section there are two possible hypotheses and one possible null hypothesis.

- **H0**: No correlation was found between exposure to an electronic serious game and understanding of introductory concepts.
- **H1**: A correlation was found between exposure to an electronic serious game and understanding of introductory concepts, given the additional presence of a standard introductory programming course.
- **H2**: A correlation was found between exposure to an electronic serious game and understanding of introductory concepts, given no other directly relevant stimuli.

1.2.3 Objectives

This research concerns itself with the use of computer games as a medium for conveying the core principles of programming. To achieve this exploration requires the completion of two intertwined objectives.

The first objective is the design and development of a game aimed at conveying of introductory principles of computer programming. This game has to meet the following requirements:

1. The game must convey basic programming concepts as described in Chapter 4.
2. The game must be understandable to users without any outside input.
3. The game must be free of any technical issues which would cause it to crash.
4. The game must not contain any conceptual fallacies which could potentially corrupt the concepts it seeks to convey.
5. The game must run on a relatively wide array of computers, so that its target audience can easily access it.

The second objective is the testing of the game so as to measure its success. This section of the research can be split into the following subsections:

1. The location of a group of people on which to test the effectiveness of the game as a tool for conveying the above mentioned concepts.
2. The identification and gathering of any relevant data on these individuals.
3. If necessary, the development of a measurable testing mechanic that can be given to those involved in this experiment to test their programming knowledge over time.
4. The administration of any testing procedure.
5. The processing and analysis of any gathered data.

1.3 Scope and Limitations

Note that the academic subject areas of education, cognitive science, learning, as well as various fields in visualisation are outside of the scope of this research. This research makes use of an electronic serious game as a method of knowledge transfer and quantitatively compares it an accepted classroom-based approach.

1.4 Thesis Document Structure

Following this introductory chapter this thesis will discuss a selection of prominent examples in the fields of gamification and serious gaming. A special focus is given

to the use of gamification and serious gaming techniques in the fields of education, especially that of computer programming.

Chapter 3 will discuss the design and development of the serious game, followed by chapter 4 which explores the design of the testing mechanisms used as well as the rationale for the sampling of those students who took part in the experiment.

Chapter 5 analyses the data gathered through the tests. Finally the conclusion will consider the results gathered as a whole in chapter 6, describe any perceived shortcomings and list recommendations for future areas of research.

Chapter 2

Literature Review

2.1 Gamification

Gamification, also called gameful design, is the application of game design techniques to non-gaming systems. Over the last three decades the gaming industry has honed the art of creating engagement between players of games and the games themselves. As gamification expert Gabe Zichermann said in a presentation to Google staff members, "Games are the only force on Earth that can get people to willingly do something they do not want to, without the use of force."(Zicherman, 2010)

Games have now been part of mainstream culture for multiple generations. Students today have more interesting worlds and opportunities available to them than those of their parent and grandparents. These worlds typically exist as entertainment and distraction and the average student may be more engaged with virtual game worlds than with everyday life. They may have become used to being engaged with these worlds and implicitly expect the same from their educational environments. This research looks to these virtual worlds of entertainment as a source of inspiration on how to engage students and learners in their educational environment.

Gamification typically involves the application of common game design mechanics to non-gaming systems (Small Business Labs, 2012), which this research applies to education. These mechanics are myriad and include leaderboards, point systems, achievement badges and ranking systems (Hayden, 2011). While these systems have been shown to produce results, they are missing the 'play' part of gameplay (Entis, 2011).

Figure 2.1: An example of Fitocracy app, showing progress bars and quests.

Fitocracy

Fitocracy is a website and smartphone application aimed at helping people keep to their exercise regiments (Fitocracy Home Page, 2012). It awards points and achievements, and publishes milestones to an online social network of the user's friends. This combination of gamification and competitiveness has garnered it a serious user base. It also showcases that weaknesses occur when gamification is applied to a non gaming system. For example, in order to gather exercise data about their users, Fitocracy requires individuals to log their own exercise sessions. This allows opportunity for players to lie and cheat the system. However, the social element, combined with the nature of exercise, might negate the risk of cheating.

Galaxy Zoo

Galaxy Zoo is an example of gamified crowd-sourcing. On the Galaxy Zoo website, users participate in the classification of deep space photographs of individual astronomical bodies (Galaxy Zoo Home Page, 2012). Users sort these photographs based on specific characteristics, such as a galaxy appearing spherical, elliptical, or spiral. In this way the program gathers information that is very difficult for computers to gather through conventional computing. The creators can ensure that each photograph is classified multiple times by building into the system re-

Figure 2.2: An example of Galaxy Zoo website, showing classification in progress.

dundancies that allow users to verify one another's classifications. This enables the system to accurately classify objects by sourcing multiple user classifications.

Stack Overflow

Stack Overflow is a popular web forum where users discuss various topics related to programming. It has more than 4 million registered users (Stack Overflow User Page, 2014). Users can vote on any topic or answer if the find it interesting or specifically useful. The user who provided the question or answer is then awarded a score related to the number of votes on their answer. A users score is displayed whenever he makes a post, allowing readers to easily see who is, and who is not a valued source of information (Stack Overflow Website, 2013).

2.1.1 Education

The following sections cover examples of gamification applied specifically to projects or products that deal with the teaching of programming or computer science skills.

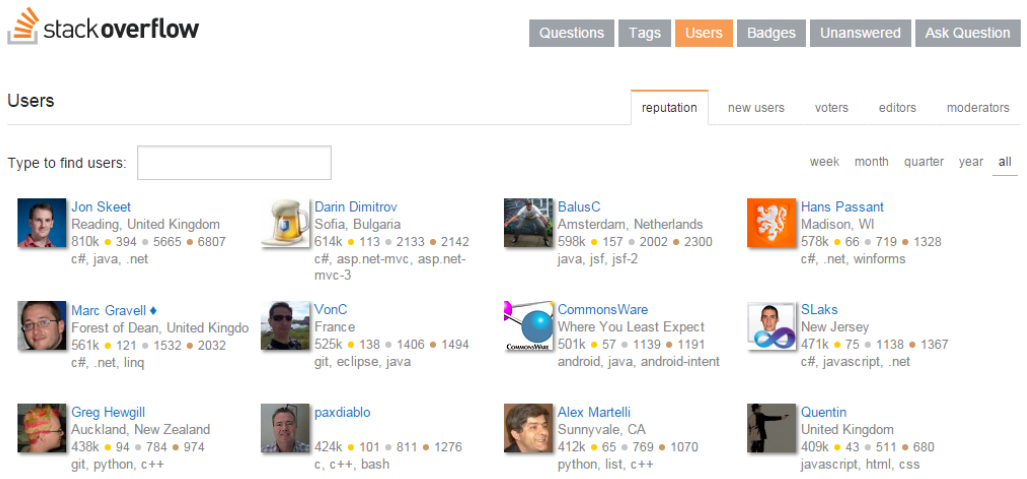

Figure 2.3: An example of the Stack Overflow reputation system.

Scratch

Scratch is a graphical programming tool aimed at a novice audience (Scratch Home Page, 2012). It was developed by the MIT Media Lab with the objective of making an easy tool that could be given to students with no programming experience. It achieves this goal by representing programming logic as interlocking "puzzle pieces", instead of traditional text-based code. These pieces join together to form logical pieces of programming code. The puzzle pieces are colour coded according to their function and are shaped in such a way that they can only join in specific ways. For example, a triangular boolean logic piece fits the triangular gap left open in a conditional statement block. Additionally, Scratch includes an animation library by default. In this way, not only do students code using a graphical interface, but because the effects of their coding is shown in a graphical, animated fashion, it becomes much easier for students to determine the results of their code.

Snap!

Snap! is an advanced offshoot of Scratch and aims to extend it to a more mature audience by adding functionality such as object-orientation, to allow for a wider variety and greater scale of programs to be developed. Snap! maintains Scratch's graphics-based user interface. One of the goals of Snap! is to make Scratch a viable language for building small applications, making for a smoother transition between Scratch and a more traditional programming language (BYOB Home Page, 2012).

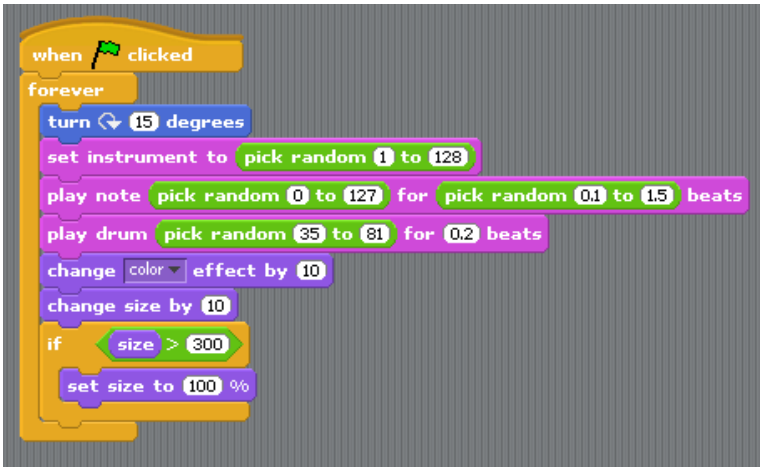

Figure 2.4: An example of a piece of Scratch code.

Codecademy

Codecademy is web-based attempt at gamifying the learning process for introductory computer programming. Codecademy uses interactive online lessons coupled with social networking and badge rewards to teach users to program using a scripting language presented in the browser (Codecademy Home Page, 2012). The use of these badges and points give users, when comparing themselves to their friends, motivation to improve themselves. Codecademy provides courses on a multitude of programming languages and concepts, so students can study through a wide range of skill levels. Each of these courses contain their own relevant sets of gamification objectives and badges.

Quest to Learn

One notable attempt made at a completely gamified school system is Quest to Learn, a high-school based in New York City. Quest to Learn is aimed at grade 6-12 learners and opened in 2009 with its first class of 6th graders. A new class is added each year. Its first cohort of students will graduate in 2015 (Quest to Learn Home Page, 2012). All aspects of the school were designed by a team of educators and game designers to maximise engagement with students. Quest to Learn shows an attempt to use gamification on a large scale within education, and that gamifying education as a whole is being seriously explored.

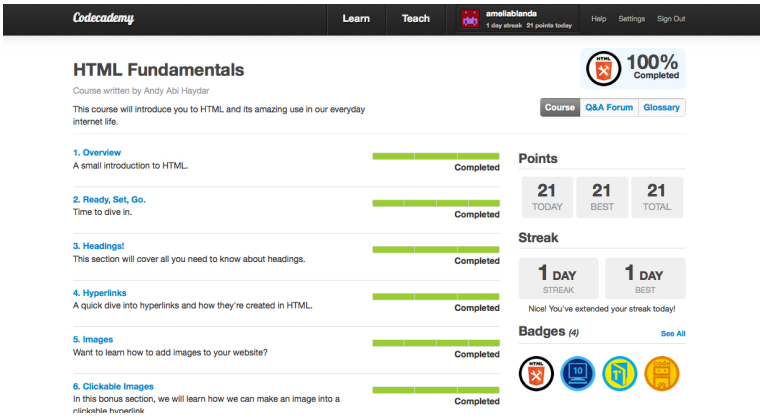

Figure 2.5: An example of Codecademy quests.

2.2 Serious Gaming

Serious games have been discussed since 1970 (Abt, 1970) and have also seen application in education. They can be defined as games with a primary purpose other than entertainment, such as learning (Derryberry, 2007). The difference between serious games and gamified systems is that serious games are whole game systems, whereas a gamified non-game system may incorporate only some elements from game design. Given this, gamified systems may or may not, for example, include actual gameplay. Serious games, on the other hand, resemble recognisable electronic or traditional games, and should ideally provide intrinsic gameplay-based reasons for users to want to engage with the system.

Ribbon Hero 2

Ribbon Hero 2 is a plug-in for the Microsoft Office suite of programs. In the game, players travel through time with Clippy, the digital office assistant featured in the same suite (Ribbon Hero 2 Home Page, 2013). Throughout their journey, players are tasked with completing certain objectives that mirror typical use of the program suite. The game is played entirely within the Microsoft Office environment, allowing users to familiarise themselves not only with concepts through gameplay, but also with ways to implement these concepts with ways to appropriately implement these concepts.

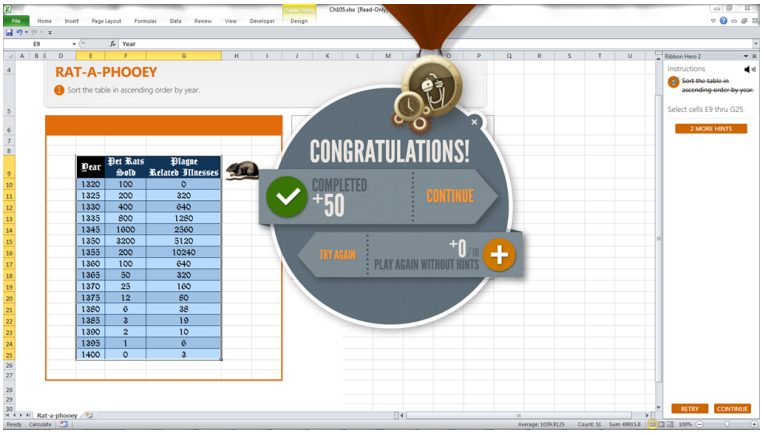

Figure 2.6: An example of gamification in Ribbon Hero 2.

Figure 2.7: Promotional material for America's Army.

America's Army

America's Army is a first person shooter (FPS) game developed in-house by the United States Army. Iterations of the game are released online for free as a public relations effort. The primary goal of the game is to show the American army in a positive light internationally. Additionally, America's Army potentially directs players to recruitment pages for the United States army, thereby fulfilling a secondary promotional goal (America's Army Home Page, 2013).

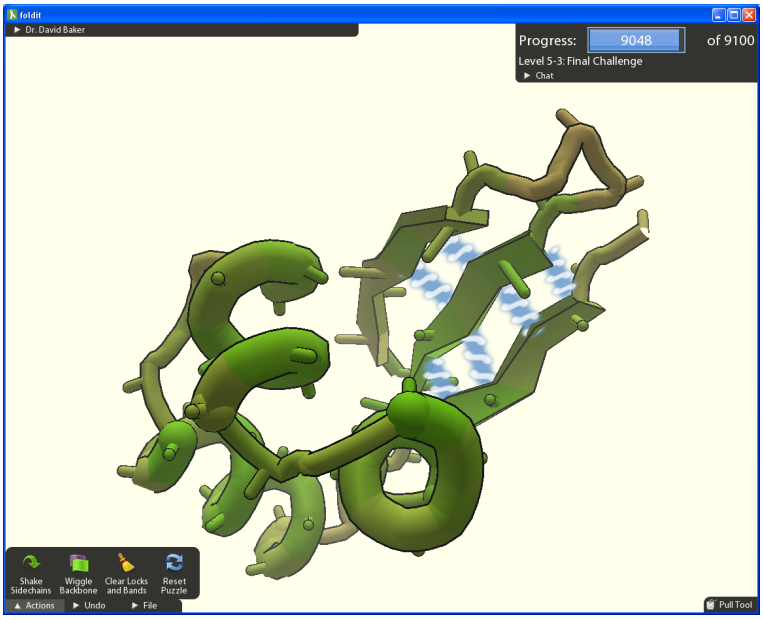

Figure 2.8: A game of Foldit being played.

Fold-it

Foldit is a game about protein folding, in which players attempt to create accurate protein structure models. Protein folding describes a range of biological excercises whereby the complex structures of proteins must be determined to understand their reactions. Due to the number of permutations inherent in the folding an individual protein, it has proven to be a difficult problem to solve using modern computational techniques (Unger & Moult, 1993).

In Foldit, problems are presented to a multitude of players who can collaborate or compete to create the best solution. Predefined metrics evaluate the problem and presents the player with a point value based on their success. This allows for multiple solutions to a single problem. For certain types of hard protein-folding problems, Foldit puzzles have produced better results than state-of-the-art computer-based solutions (Khatib et al., 2011). Data gathered using Foldit is being used to develop cures for HIV/Aids, cancer and Alzheimer's disease amongst others (Foldit Science Page, 2013).

Figure 2.9: A game of Ingress being played.

Google Ingress

Google Ingress an augmented reality game. Augmented reality is the act of overlaying digital information on the physical world through the use of some digital medium, such as smart phones or tablets (Carmigniani et al., 2011). Augmented reality games can be characterised as using input from the real world to determine output.

In the case of Ingress, players visit real-world points of interest, where they must engage with and fight alien invaders with the help of other players (Ingress Home Page, 2013). While this game mechanically follows established game design patterns, Google can use it to gather foot traffic data for their map applications. Additionally, since the points of interest are tagged by users, Google can also incorporate those into its maps.

2.2.1 Education

In contrast to section 2.1.1 , the next section specifically looks at serious games dealing with the education of programming or computer science skills.

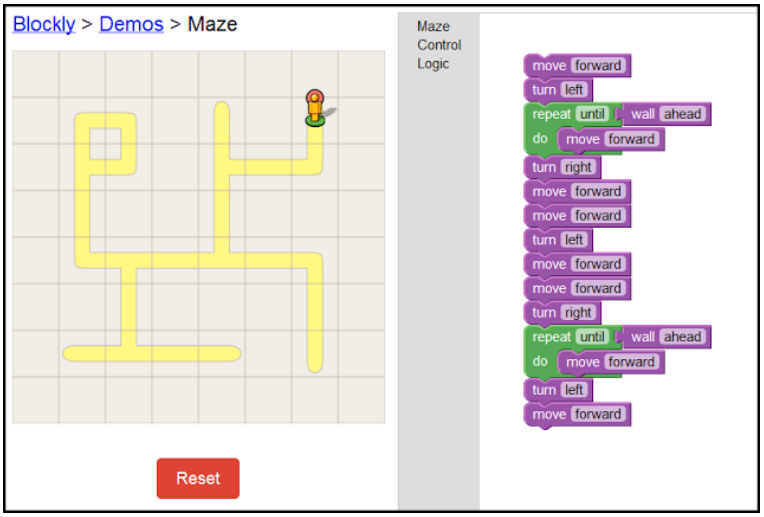

Figure 2.10: A Google Blocky maze navigation puzzle with code on the right.

Google Blocky

Google Blocky is another visual programming tool, similar to Scratch, in which segments of coding logic are snapped together. However unlike, Scratch, Blocky takes a serious game approach to learning. Players are asked to solve puzzles using the blocky interface, such as guiding a robot through a maze, or drawing visuals in a fashion similar to the turtle in the Logo programming language (Logo Foundation Home Page, 2013). Additionally, Blocky can automatically generate Javascript, Python and XML code based on the visual programs users create (Google Blocky Home Page, 2013).

Lightbot

Another example of a visual code snapping programming serious game can be found in the browser-based flash game Lightbot and its sequel Lightbot 2. These two games are short puzzle games that make use of programming-style logic to navigate a small robot around the game world. Although not developed as teaching tools, the games make use of conditions, functions and recursion as part of game-play (Yaroslavski, 2012). Lightbot also completely abstracts away from code based programs instead makes use of symbols and a drag-and-drop interface, thus conceptually opening up the game to a wider audience.

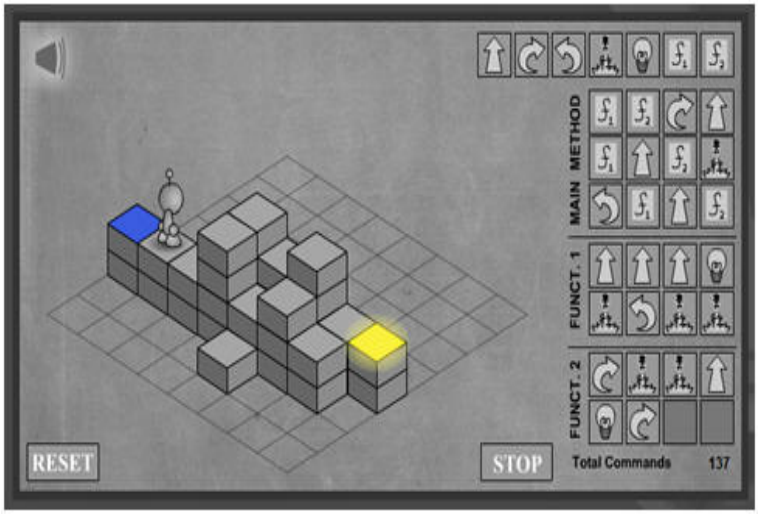

Figure 2.11: A puzzle in Lightbot 2, featuring player controlled instructions on the right.

Alice

Alice, a graphical programming language, is aimed at building interest in programming amongst young women focusing on storytelling and animation. Alice's focus on storytelling, specifically its users' ability to create their own stories' proved to increase user interest with the tool over a version without a storytelling focus (Kelleher et al., 2007). This suggests that personalising the goals and outcomes of programming problems to a specific target audience could lead to increased interest in the field amongst that audience. Studies making use of Alice have produced results showing rises in user interest, but are vague on the skill transference abilities of such programs.

2.2.2 Marques, 2013

In a game developed at the University of Witwatersrand in South Africa, users use an interface with a custom programming language to direct traffic (Marques et al., 2012). Each level consists of a tunnel from which a succession of vehicles is generated. Each vehicle is one of six colours (red, blue, white, etc.), and one of several types (bus, car, etc.). Vehicles drive towards intersections. These intersections must be programmed by the player using if-else statements and Boolean logic to guide cars to predetermined garages. In this fashion, players of the game can be

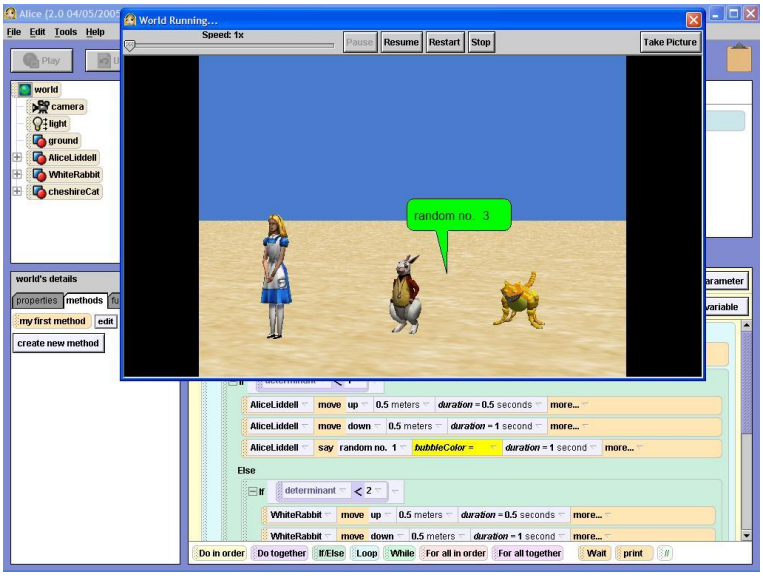

Figure 2.12: An animation scene being coded in Alice.

taught about relatively complex conditional statements and embedded conditional statements.

Unless the game is paused, vehicles will start behaving according to the newly coded rules for an intersection as players code. This allows players to immediately see the effects of any code written. It also allows the players to see the basic flow of the program at any point in time. This is especially useful given the easily identifiable colour/model combination of the vehicles.

Marques, the designer of the game, tested various versions of his prototype with three volunteer groups from various programming backgrounds, with some having significant programming experience and others little to none. The students were rated using metrics derived from the game play itself, such as number of lines of code used and time taken to complete the level. While translating these metrics into something resembling "programming skill" is certainly questionable, Marques also referenced the average class mark of these groups and compared the marks with those in the class who were not part of the study. While most of the groups did not show a discernible difference in averages the group who had no previous programming experience showed a 7% increase over those in the class who did not take part in the study. However, it is important to note that no additional statistics, such as the level of statistical significance or standard deviation, were provided in the study. Nonetheless, the study does indicate the viability of a serious game approach.

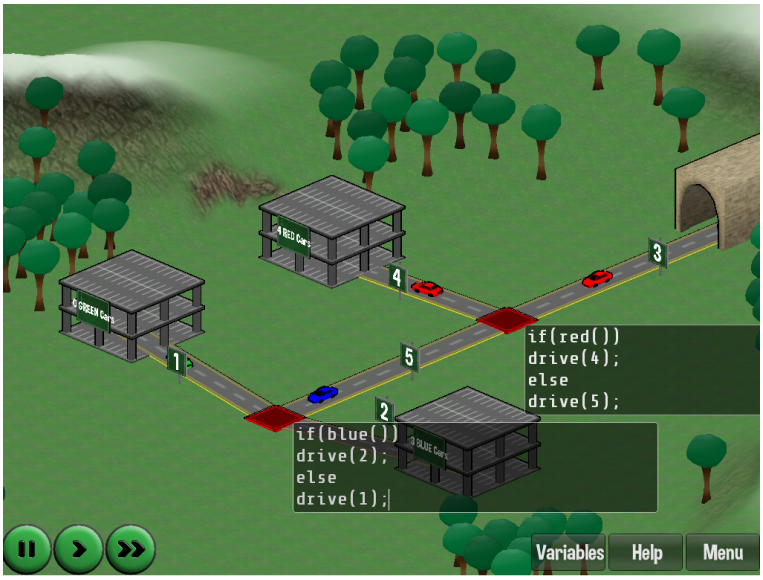

Figure 2.13: A serious game developed at the University of Witwatersrand, showing code at traffic intersections.

Current attempts at the problem

Several new commercial and academic serious game approaches to computer language learning through games are in development.

Code Hero makes use of a first-person viewpoint and lets players solve puzzles by using a "code gun" that shoots user defined code at items within a predefined game world (Code Hero Project Page, 2012)). Another new project is "else{Heart.break()}", in which the player controls a character in a player-reprogrammable world. As the story of the game unfolds the player is taught to code by in-game characters and more of the game's code becomes mutable (else{Heart.break()} Home Page, 2012).

CodeSpells is an RPG in development at the University of California in which players use a custom programming language to write magic spells to use on objects within the game world(CodeSpells Game Page, 2013).

Lastly, BotLogic is a mobile game in development that is aimed at teaching young children procedural logic and algorithms through a drag-and-drop interface, similar to the approach presented by Lightbot(BotLogic Home Page, 2013).

While none of these games have been fully released, the presence of new additions

to the field indicate that there is significant interest by both academia and industry in trying to teach programming and its associated skills through gameplay.

2.3 Summary

This chapter discussed various general, and specifically educational, gamification and serious gaming implementations, their purposes, design decisions and implementations. These examples of previous work were used to guide the creation of the serious game developed to achieve the goals of the research. Chapter 3 brings together the mechanical system and puzzle design decisions in the game that resulted from these guidelines, and also provides a comprehensive description of the systems. Specific focus was given to the previous work done in programming education through serious games and the creation of a novel design, namely the use of symbols to represent programing logic instead of text. This distinction is described in full detail in section 3.2.3 of chapter 3.

Chapter 3

Game Design and Development

3.1 Introduction

This chapter covers the design and development of the serious game developed to test the hypotheses discussed in Chapter 1. The research had the following objectives for the game design:

1. *The game must convey a coherent set of the basic programming concepts.* If the game fails to meet this objective it will not be usable as an educational tool for either of the hypotheses. This objective informs the core of the game design.

2. *The game must be understandable to users without any outside input.* In order to avoid potential external contamination of the concepts the game seeks to convey, it must communicate them within the game.

3. *The game must be free of any technical issues which would cause it to crash.* Any technical issues that render the game unplayable or frustrates the player must be avoided to avoid respondents from dropping off and reducing the data set available for testing.

4. *The game must not contain any fallacies which could potentially corrupt the concepts it seeks to convey.* The game must convey programming concepts accurately in order to allow for a valid comparison to traditional class-taught programmers.

5. *The game must run on a relatively wide array of computers, so that its target*

audience can easily access it. This is important to maximize the amount of data gathered during the testing period.

3.2 Game Design Background

Various factors and fields were used as a foundation in the design of the game and are discussed in the section.

3.2.1 Programming Concept Focus Area

Opinions differ on what qualifies as introductory programming concepts. While most experts agree on which topics should be included in a typical introductory programming textbook, it is the order in which these topics are introduced tend to differ.

It was decided to focus on an isolated subset of introductory programming concepts in order to allow the topics to be conveyed as coherently as possible, without any prerequisite knowledge. In addition, the available development time for the experiment had to be taken into consideration.

An analysis was performed on several contemporary introductory textbooks to determine a focus area for the prototype. These included the two textbooks used by the Department of Information Science, which were provided to respondents as their introductory textbooks during this study.

The content of the textbooks on a per-chapter basis by comparing the order in which the following concepts were introduced:

1. *Variables*: The concept that information can be tied to certain variables and manipulated abstractly.
2. *Conditional Statements*: The concept that a program can act differently based on defined criteria being met.
3. *Loop Structures*: The concept that code segments can be repeated, allowing for greater control of programming flow.
4. *Data Structures*: More complex forms of data storage and the manipulations of these structures. Includes arrays, lists, tuples, etc.

5. *Methods*: The abstraction of sections of code into easily reusable segments.
6. *Classes*: Any discussion of object orientated programming and the practical implementation of these principles.
7. *Files*: Any discussion of the use of files for input and output.

While the names of the concepts differ between textbooks and languages, the analysis qualitatively determined when the concepts were introduced for each textbook.

The above analysis is presented below in Table 3.1.

It was decided to focus on the basic concepts of variables, conditional statements and loops, as it is clear from the analysis presented in Table 3.1 that a majority of the analysed works agree that these concepts should be introduced first and in sequence. It is only after the introduction of these concepts that the textbooks start to diverge. In addition, these concepts blend well and provide a basis for coherent game design.

3.2.2 Visualisation

Significant work has been done regarding the visualisation of various domains related to computer programming. Work has been done in visualising program source structures (Caudwell, 2009), data and information (Tufte & Graves-Morris, 1983), and human-computer interfaces (Raskin, 2000; Crawford, 2002) amongst many other fields. While elements of all these fields can be tied into the topic of this thesis, its primary concern is with the visualisation of program logic for the purpose of explanation to a novice audience.

3.2.3 Text- versus Symbol-Based Code Representation

As demonstrated in Chapter 2, some work has been done in the field of interactive educational programming games. However, the majority of these games make use of a coding user interface, meaning players interact with the game world or puzzle by means of writing some sort of codified language. In some cases this language is similar to a programming language, such as Java, while in others the language is game-specific and does not mimic an existing language. Regardless, the majority of attempts at making an educational programming serious game used text coding as the medium of interaction (Kelleher et al., 2007; Marques et al., 2012; Google Blocky Home Page, 2013; CodeSpells Game Page, 2013).

Table 3.1: The order of concept introduction in a selection of textbooks.

Microsoft Visual C# 2012 An Introduction to Object-Orientated Programming (Farrell, 2012)	**Java Actually: A comprehensive primer in programming (Mughal et al., 2008)**
Variables	Variables
Conditional Statements	Conditional Statements
Loops	Loops
Data Structures	Data Structures
Methods	Classes
Classes	Methods
Files	Files
A beginner's guide to Programming Logic and Design: Comprehensive (Farrel, 2013)	**Introduction to Python Programming and Developing GUI Applications with PyQ(Harwani, 2012)**
Variables	Variables
Conditional Statements	Conditional Statements
Loops	Loops
Data Structures	Data Structures
Files	Methods
Methods	Classes
Classes	Files
Python Programming: An Introduction to computer science 2nd edition(Zelle, 2010)	**Fundamentals of Python: From first programs through data structures(Lambert, 2011)**
Variables	Variables
Loops	Loops
Methods	Conditional Statements
Objects	Files
Data Structures	Data Structures
Conditional Statements	Methods
Loops	Classes
Fundamentals of Programming using Java(Currie, 2006)	
Variables	
Decision Statements	
Loops	
Methods	
Data Structures	
Files	
Classes	

Academic programming courses typically make use of a specific formal coding language as a medium to teach programming. Some attempts have been made at textbooks that teach programming concepts without teaching a specific programming language Farrel (2013).

Non-coding approaches are much more rare. One such approach makes use of symbols to represent code. Here commands are represented as graphical symbols in some fashion. In this way, an arrow symbol could command an on-screen robot to walk ten steps, which in turn could be looped through the use of some looping symbol (Yaroslavski, 2012; Manufactoria Game Page, 2013).

A coding approach would conceivably lend itself into easier integration with actual coding practices, as it could focus on the syntax and definition of coding, whereas a symbol-based approach could forgo coding specifics and instead focus on problem analysis and problem solving in programming.

Significant research attempts have been made in text-based coding serious games (CodeSpells Game Page, 2013; Muratet et al., 2010), most of which have shown promising results regarding their effectiveness as learning tools (Kelleher et al., 2007; Marques et al., 2012). On the other hand most of the attempts at symbol-based code representations have been small games made primarily as entertainment (Yaroslavski, 2012; Manufactoria Game Page, 2013), and not for pedagogical purposes. This indicates that people are willing to play symbol-based games for entertainment and of their own volition. As such, a symbol-based approach was used in the prototype developed as part of this research. Positive results from tests with such a game may be comparable to the results of previous text-based approaches and would open up further avenues for the development of an entertaining educational game.

3.2.4 Process Flow Diagrams

A process flow diagram is a programming design too used to graphically represent an algorithm or program. One style of process flow diagrams is the Unified Modelling Language (UML) set of program design and development tools. For this reason, process flow diagrams provided useful inspiration in designing a game aimed at teaching programming concepts while avoiding the use of programming language.

Process flow diagrams, or activity diagrams, make use of symbols to represent code structure. The procedure or flow of an algorithm is represented by unidirectional arrows between these symbols. The symbols typically have one arrow pointing

toward them and one arrow pointing from them. An executed command such as "Increase counter by 1" is typically shown in a square box. A diamond typically represents a decision such as "Is counter greater than 10?" and allows for multiple arrows moving outward. Through these basic symbols, as well as arrows that point back to earlier symbols, process flow diagram, can represent the basic programming concepts of iteration and conditional branching.

Figure 3.1 shows a simple activity diagram describing the login process for a website. Note the arrows indicating the logical order of the process, as well as the diamond checking if a user has entered correct login details.

Process flow diagrams are often a preliminary step in writing code, because it allows the programmer to focus on the logic, also called semantics, of a problem, rather than the coding, also called syntax, itself.

Because of this focus on programming logic over programming language, process flow diagrams were the inspiration for representing algorithmic or procedural thinking processes in-game to non-programmers. Through this inspiration, the game avoids the use of a language-based approach for programming code.

3.3 Design Decisions

This section discusses the development of the research prototype.

3.3.1 Target Users

The game is aimed at introductory level students, ranging from children in primary or high school to university or adult students learning programming for the first time. Specifically, it is aimed at students who are having difficulties understanding the underlying logic of introductory level coding, as opposed to the writing of the code itself.

3.3.2 Focus Areas of the Game

The prototype focuses on demonstrating the three core ideas in entry level programming described earlier in this chapter, namely 1) variables and mutability, and 2) process iteration, 3) process branching. As discussed, in a typical introductory programming text book these concepts would be covered in the first few

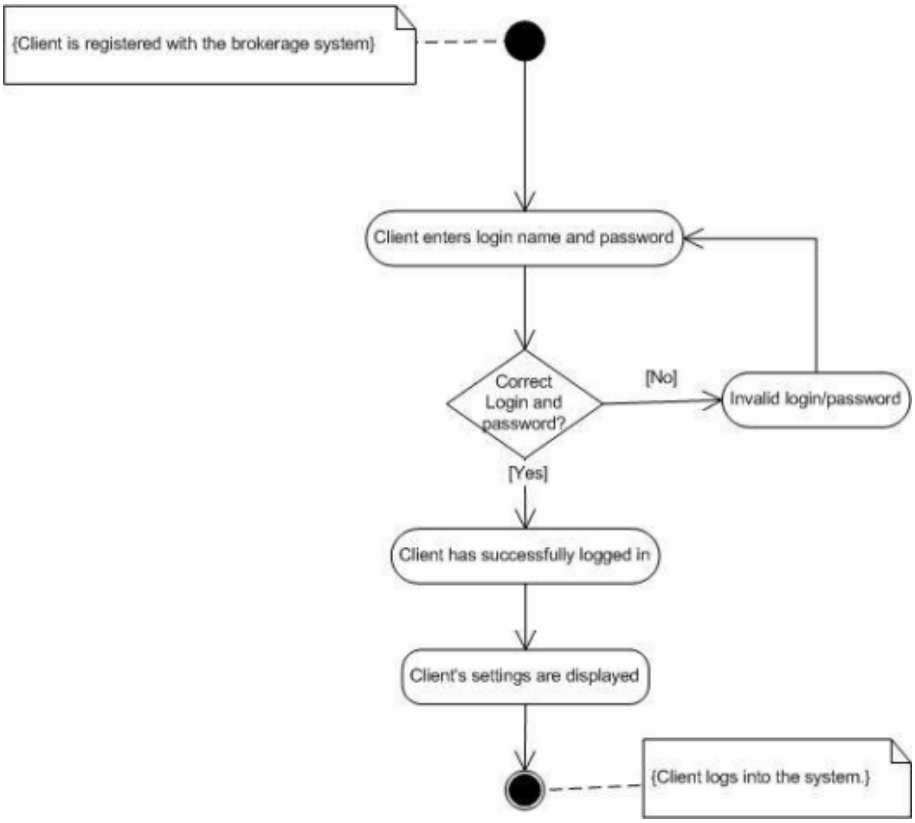

Figure 3.1: An example of an activity diagram showing a high level website login algorithm (Lucidchart - Activity Diagram Introduction Page, 2015).

chapters by discussing concepts such as variables, operators, for- and while-loops as well as if-else statements.

Players are gradually introduced to these concepts through a series of puzzles, first incorporating only some and later all of the elements described above. Additionally, the difficulty of these puzzles are increased over time, leading the player firstly to focus on the initial conceptualisation of solutions to the problems and, secondly, allowing more opportunity for feedback from the game through the solving process.

Overall, the game consists of fifty-five distinct game levels or puzzles split into eight segments, each focusing on different areas of the game. Three of the eight sections are tutorial segments, that introduce players to concepts through on-screen text and prompts. These tutorial segments comprised fourteen of the fifty-five total levels.

3.3.3 Game Mechanics

The player directs an object, called a carrier, by moving it through various level that contain objects with which the carrier interact. Most notably, the carrier is used to pick up and move gems, manipulate them, and drop them in goals. When enough gems have been dropped in goals, the player has completed the level.

The next section details the mechanics of the game, explaining their relevance to gameplay and, where appropriate, indicating where specific mechanics act as analogues for programming concepts.

The Level

Each level consists of a two dimensional grid of objects or symbols. Each cell in the grid may contain one object. Each of the symbols contained in these cells represent a specific interaction, similar to symbols in process flow diagrams.

The Carrier and Flow

The carrier is the only moving object in the game. Upon the initialisation of the level, the carrier is placed on a specific launcher that launches it in one of the four orthogonal directions (up, right, down or left).

Additionally, the path that the carrier is allowed to take is highlighted with flow lines. These lines, combined with the instructional symbols, to present a picture similar to that of a typical process flow diagram.

Figure 3.2: The carrier

Build Time and Run Time

The game is split in two phases. During the build phase the player can place symbols on the level board to direct the actions of the carrier. The player can use this phase to move, change or delete objects placed on the board during previous

build phases. By default, objects placed as part of the level cannot be moved, changed or destroyed. The carrier does not move during the build phase.

When the player activates the run phase, the carrier begins moving in the direction dictated by the carrier launcher objects. During this phase the player cannot move, alter or delete any of the placed symbols. Ideally, the player would have solved the level and the carrier would run to completion, causing the game to open the next level.

However, if the carrier runs into a situation that causes an error, or the run time is cancelled by the player, the game will revert to the build phase. This will reset all variables and counters to their initial values. Essentially, the level returns to its default state, with the exception of any symbols placed by the user.

These two phases are designed to reflect the iterative process of design, development and observation that programmers use when developing software.

Arrows and Direction

When the carrier objects collides with arrows, it changes direction and goes either up, down, left or right. Additionally, if a placed arrow objects intersects a process flow line, it will reflect the new direction the carrier will take during the run phase. The arrows only point in the four orthogonal directions. Diagonal movement is not used in the game.

Each symbol can be interacted with on two paths due to the orthogonal nature of the carrier movement, as well as the two dimensional nature of the level board and emergent factor of gameplay incorporated into the level design. A symbol can be crossed from left-to-right as well as top-to-bottom. This leads to dynamic level design when combined with mechanics designed to limit the players decision space. These will be discussed later.

Figure 3.3: An arrow pointing upwards.

Gems

Gems represent variables in the game. When a carrier moves over a gem it will pick it up, which causes the gem to move with the carrier. In this way, the player now has focus on that gem and can manipulate it and other symbols. Gems can be one of five incremental sizes. Triangular gems are the smallest, followed by squares, pentagons, hexagons and, finally, circular gems. Additionally, gems are one of three colours, namely orange, green or blue. Both variables (number of sides and colour) can change by placing specific objects on the level board.

Figure 3.4: A full series of red gems

Changing Gem Values

Players have access to two objects that can change the size of a gem. The plus symbol will increase a gem's size category by one, and a minus symbol will decrease it by one. Incrementing the largest gem, a circular gem, or decreasing the smallest gem, a triangular gem, counts as a runtime error and will cause the game to switch from the run phase to the build phase, thereby reseting a player's progress on that specific level.

Colour change symbols are also present in the game. Unlike the increment and decrement symbols however, colour change symbols cannot be placed by players, and are only present if they form part of the puzzle the players must to solve. This was a design decision made to force players into certain desired puzzle solutions. Colour change symbols only change a specific colour of gem into another predetermined colour.

The three symbols (plus, minus, and colour change) can be seen as representing mutator methods, if the gems represented objects. They can also represent operators, if the gems are seen as variables.

Figure 3.5: An incremental, decremental and colour change symbol.

Gem Spawners and Goals

Gem spawners are objects that spawn the gems the player moves around using the carrier. Conversely, goals are the objects a carrier carrying a gem interacts in order to deposit it and progress towards completing the current level.

Gem spawners generate a specific sequence of gems in a specific order. Once a gem is dropped off at a goal, the next gem in the sequence is created at the spawner position. This sequence consists of up to fifteen distinct symbols, each of which has an arbitrary level and colour.

Similarly, each goal only accepts as valid only certain gem level-colour combinations as valid. A gem is removed from the board, the score increased and a new gem generated at the spawner of the previous gem only if a valid gem is brought into the same cell as a goal accepting that colour and level combination.

The concept grounding the arbitrary sequences of gems is to simulate random algorithmic inputs. When dealing with input outside of the control of the program, a programmer must take into account a wide variety of potential inputs.

Figure 3.6: Spawner and goal.

Split Symbols

Split symbols are the games conditional statements and would be equivalent to simple if statements in a programming language. Each split symbol can be given one level value and one colour value, matching a specific gem.

If a split symbol is placed on a horizontal flow line, it will create a secondary flow line leading downwards. this causes a split in the line and indicates to the player that the carrier could take one of two paths at that point during runtime.

Figure 3.7: A split symbol checking for green triangles.

It is important to note at this point that it would have been possible to use fairly complicated boolean logic check to build puzzles which the players must solve. However, since the focus of the game is to convey the concepts of program flow and not boolean logic, it was decided to limit the puzzles to relatively simplistic boolean checks. As it is implemented in the game, the split symbols will check for only one type of gem and send it in a downward direction if a carrier containing a matching gem passes over the split symbol.

Loops

A typical programming language provides three types of loop functions. Repeat loops, as the name suggests, are used to repeat a section of code for a predefined number of times. A while loop is used to repeat a section of code until a specific condition is met. Lastly, a for loop is used to repeat a piece of code a specific number of times, but has a built-in variable that changes in a predefined way for each iteration. While these loops are alike and can be made to fit most situations, they are distinct concepts to inexperienced programmers and are taught as such.

In the game loops can be set up by using a mimum of four arrow symbols pointing to one another. Players can use loops in conjuction with split symbols to set up the equivalent of a while loop.

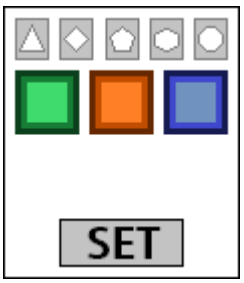

Figure 3.8: A split symbol customisation menu.

Figure 3.8 shows the menu for customising split symbols. The first row sets the desired gem size. The second row sets the desired gem colour.

Walls

Walls were implemented to limit player decision making, or to force players to consider sets of pre-placed symbols in a specific sequence. A wall is an object, taking up one cell, that does not allow the carrier to pass over it. If a carrier collides with a wall, it counts as a runtime error. This resets the level and puts the player back into build mode. Any flow lines also end when colliding with a wall.

Figure 3.9: A wall

Limiting Player Options

Player decisions can be limited in three ways on a per-level basis in order to introduce concepts in a controlled fashion, as well as give more options for level designs. Firstly, player access to specific symbols can be restricted to remove certain functionality. With access to all of the symbols, several of the puzzles can be solved conceptually in a number of different ways, so this restriction is also useful when encouraging a particular mindset.

Secondly, a limit on the number of symbols per level can be defined. Each placed symbol counts towards a running total. If the total is met, a new symbol cannot be placed until a previous symbol is deleted.

Thirdly, walls and other pre-placed symbols can limit a player's choices in any situation, which is useful in reinforcing a desired thought pattern or when teaching new concepts or new uses for familiar concepts.

3.3.4 Technical Specifications

The game was developed using Python 2.7.2, as well the pygame 1.9 library for Python. Pygame was used for the drawing logic, as well as other gameplay logic, such as collision detection.

Other game development platforms were also considered. Game Maker(Yo Yo Games, 2012) is a popular 2D game creation kit but was rejected as it does not

Table 3.2: Comparison between game mechanics and programming concepts

Game Mechanic	**Programming Concept**	
Carrier	State	
Build time vs. running time	Code Compilation	
Arrows	Sequentiality	
Gems	Variables and objects	
Gem spawners	Unreliable user input	
Loops created using arrows	Code iteration	
Split symbols	Conditional statements	
Walls	Separation of complex logic	

allow access to it's entire code base, particularly the low level game loop code. Unity is a another platform that was considered as, at the time of the projects start, Unity(Unity Home Page, 2012) only allowed for the creation of 3D games, which was considered unnecessary for the goals of this project.

This project did not concern itself with the creation of a standard for the creation of educational games, nor with any particular level of game scope scalability, and as such made use of a once-off coding architecture. Standard object-orientated practices where followed for control of in-game elements.

After a single initialisation phase, the game made use of an infinite loop to iterate over three phases in the program. The first of these listened for keyboard and mouse input from the player and passed any caught input events to the relevant object. The second phase performed the game logic, resulting in state changes for the game objects. The final phase updated the rendering logic on a specific frame count.

3.3.5 Level Description

Tutorial 1: Basics

The introductory tutorial introduces the basics concepts of the game and actions such as placing objects, impacting flow, moving gems and dropping gems. Players are also introduced to basic loops.

Movers

Movers is the first level set players must complete. In this level set, puzzles revolve around players picking up and dropping off gems in the correct sequence. Some puzzles involve loops and others only a set of instructions.

Tutorial 2: Change

In the second tutorial players are taught that gems are mutable. They are first introduced to the concepts of gem levels, plus and minus symbols, and colour change symbols. Players are also shown that a level will reset if gems grow or shrink beyond their constraints. This tutorial also teaches players how to place plus and minus symbols, and that colour swap symbols are static and cannot be placed by the player.

Warm-up

The second set of levels players must solve includes a mixture of game elements. Some levels require players to simply move over a set of symbols in the correct sequence; while others require players to use the same symbols multiple times, necessitating them to plan the order in which certain parts of the puzzle is approached. Other puzzles are unsolvable without the use of individual symbols multiple times. Additionally, the puzzles vary in giving players the ability to place plus and minus symbols, or whether they use the available symbols.

All together

The third set is similar to the level sets described above but with slightly more difficult puzzles and with a focus on integrating all the elements the players have been introduced to so far in the game in each individual level. No new mechanics are introduced in this level set.

Tutorial 3: Decisions

The third tutorial introduces split symbols and the concept of decision making into game flow. Players are shown how the split symbol functions and how it can be used to direct the carrier. Players are also shown how to place, and later customise, the split symbol to fit each problem.

Choices

The fourth level set uses all the elements introduced thus far, including the split symbol. As before each element is used separately or combined in small ways, so the player may focus on one problem at a time.

Crossroads

The fifth and final level set contains complex puzzles that use all or most of the mechanics in the game. At this point, players have shown proficiency in and understanding of all the individual mechanics and are tasked with combining them in this final level set.

3.4 Summary

This chapter discussed the design and development process of the programming educational game developed for this research. It discusses the design decisions and sources of inspiration for the games mechanical systems, as well as the puzzle design for the game. This also covered half of the requirements set forth in chapter 1. The next chapter presents the development and testing of the measurement mechanics.

Chapter 4

Test and Experiment Design and Development

4.1 Measurement Design

Testing the effectiveness of the finished programming educational game prototype required a longitudinal test. Longitudinal tests take similar measurements spread over time and compare them to look for changes in specific variables. This research measures students' programming aptitude before and after two sets of stimuli, namely a traditional classroom environment as well as the game developed for this project.

The research aim is to study the comparable empirical differences between effects of the stimuli, described above, on the volunteer test groups.

Four groups of potential respondents were identified:

1. A group with no programming experience tested before and after playing the prototype game.
2. A group with no programming experience tested before and after taking a traditional class-driven introductory programming course.
3. A group with no programming experience were tested before and after taking a traditional class-driven introductory programming course as well as playing the prototype.
4. A control group, tested twice, with no stimuli. This group served to isolate

the degree to which a student's ability in taking the test improved solely through previous exposure to the test.

In order to correctly measure and differentiate between these group, several sources of data were measured. Due to the nature of the sampling process, and that respondents are university students, the marks of students' relevant course-work were taken as one data source. While this does provide an easily compatible metric, one has to consider some inherent problems of such a measure.

Coursework scores would be derived from tests or projects set on the specific work covered in such a course. For example, a class doing an introductory Python course, would assess programming logic expressed in Python, as opposed to just programming logic. Additionally, for groups 1 and 4, no such class marks would exist. Thus, while class marks would provide useful additional data it should ideally not be the only source of data used to support the second hypothesis, that of gameplay being used to completely replace course work, describe in Chapter 1.

4.1.1 Test Design

Language-Agnostic Test

In addition to class marks, a language-agnostic programming aptitude test was used for additional data. While research has been done on programming aptitude tests (Dehnadi, 2006), no standard test has emerged. Given this, a test was developed that could measure the change in a persons' programming ability, given both the stimuli of the educational game and the traditional introductory class.

The creation of a language-independent programming aptitude test falls outside the scope of the research. Thus, the research focused on determining the difference in skill between points in time, rather than determining skill at a given point at time. In this way, the average growth in skill for a given group is used to determine the effectiveness of given stimuli.

Inspired by standard IQ tests, the test developed is comprised of basic level pseudo-code programming problems. Participants are given an example problem and answer that show the basic logic of a concept and are asked three questions of increasing difficulty. Consequently, each category of questions contains one example and three questions. Categories of questions are tied to programming principles, such as variables, iteration, conditionals, etc. In total, the questionnaire contained twenty-three questions.

Example 1:

Given:

a has a starting value of 10

And the following commands:

```
if a is 15 then do:
     b ← 5
otherwise do:
     b ← 7
```

then a will have an end value of 10
and b will have and end value of 7

Question 1

Given:

a has a starting value of 2
b has a starting value of 3

And the following commands:

```
if (a+b) is 5 then do:
     a ← a - 1
     b ← b - 1
otherwise do:
     a ← a + 1
     b ← b + 1
```

What are the end values of a and b?

Figure 4.1: A section from the first language-agnostic programming test given to the students.

All questions dealt with variables with a certain starting value that change in some instructions sets, after which participants must give the end value. This minimised the number of possible answers in each question. Participants need only basic arithmetic skills as a prerequisite to correctly answer each question.

To test for growth in knowledge over time, participants wrote the test before and after each set of stimuli. The first and second test were identical in structure but with the values of the variables changed so that participants could not copy the answers across tests.

Please rate the accuracy of this statement:				
Playing the game gave me insight into the way programming languages work				
Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

Figure 4.2: An example of a 5-point Likert-scale question.

Other Measurements

Two smaller measurements were also designed to gather more descriptive data about the respondents. Firstly, two questions were included in the language agnostic test to determine the initial experience of the student respondents by asking about their level of education in the fields of mathematics and computer science.

The second descriptive test, a survey, the impression the respondents had of the game, as described in Chapter 3. This survey consisted of two questions on time spent with the system, as well as twelve five-point Likert-scale questions asking respondents about their impressions of various parts of the system. This survey can be seen in appendix C. Likert-scale questions ask respondents to rate how strongly they agree or disagree with a specific statement.

4.1.2 Testing Procedure Design

In order to homogenise the sample groups as much as possible, a large group of potential respondents had to be identified. This would need to include respondents who were to be enrolled in an introductory programming course; and respondents who would not be enrolled in such a course but could be used in comparison. University or high school learners who were going to enroll, or were already enrolled in an introductory level programming course seemed ideal. However, several restrictions had to be taken into account in the sampling procedure:

- The group must be large enough to compensate for the inherent variation of individuals. Forty respondents per test group would be ideal.
- The group must be homogeneous while programming students and non-programming students still needed to be distinguishable.
- Due to the longitudinal nature of the study, the group remain involved with the study throughout their education period.

- As the study was not officially affiliated with any specific degree program, all student respondents had to volunteer for the duration of the study.

Because of the restriction requiring respondents to volunteer for duration of the study it was decided not to target school learners. School learners typically start an introductory programming course at sixteen years old and so would be subject to greater legal and institutional restrictions. On the other hand, university students who are starting a programming course would legally be able to give permission for their own involvement.

Due to the time frame of the tests there were only a few opportunities available, typically at the start of each academic semester, for effectively testing a new group of respondents.

Because of the timing constraints inherent in the test, the subject matter being tested, as well as the administration involved in organising each potential test group, the number of viable tests in the academic year is limited to two opportunities for each academic department or group, once at the start of each semester.

Finally, respondents were financially reimbursed for their participation in order to maximise their numbers. This, however, increased the severity of any false starts, as such an occurrence severely limited the financial resources available for this research.

Due to these factors, it was decided that as many initial student respondents as possible would be the most likely to produce illuminating results.

A single introductory course in a specific academic year group was targeted and asked to participate. The students were asked to complete the developed test, after which a subset played the game throughout the course. Both groups then wrote the second test after completing their course. Both groups would provide data on the possibility of using serious games as enhancement to standard classroom processes.

A second set of students, in the same academic department as the first groups but uninvolved in programming courses, were asked to participate in order to test the second hypothesis of replacing the standard classroom process. Similar to the first two sets of respondents, these student were split in two groups: one playing the game and the other group only writing the two tests, with some time passing between, to prevent students learning the test as they write.

When considering these restrictions, as well as the sampling design, three university faculties and departments were identified as sources of potential respondents:

1. Electrical engineering students who study computer programming in the *Engineering Faculty*. All first year students, however, complete a short programming course, which would lead to exposure in the test groups and are therefore not ideal candidates.

2. The *Computer Science Division* in the *Department of Mathematical Sciences* specialises in programming education. Other students of this department would provide a potential control group.

3. Finally, the *Faculty of Art and Social Sciences* houses the *Department of Information Science*. Half of these students start programming courses in their second year which is for many of the students their first exposure to the field.

Of these three groups, the Engineering Faculty provides the least attractive potential respondent pool because all first year students are enrolled in an basic programming course during their second semester that covers much of the material the game seeks to support. This meant that there would be no group of comparable students with no programming experience, which could be used as a control group.

The second least desirable group is the Department of Mathematical Sciences. While they provide sufficiently distinct students, the formal requirements of the department make it likely that any potential respondents would have had significant previous exposure to programming or programming-like environments, meaning that the two groups could potentially test closer together.

The most desirable of the three groups is the Department of Information Sciences. These students tend to take programming as an elective, minimising any potential previous exposure to the discipline. In addition, the department has a relatively homogeneous student body in the Center for Knowledge Dynamics, with students studying Decision Making who do not have any earlier formal programming exposure.

4.2 Final Target Group

Considering the above-mentioned factors, it was decided to target second-year Socio-Informatics students as the two groups with exposure to a traditional programming class. Second year decision making students share some classes with Informatics students, typically without electing to take the programming course.

They would provide the non-programming groups. The Informatics class group were close to ninety students in 2013, whereas the Decision Making students were approximately one hundred students.

4.3 Test Deployment

Testing started in the first semester of 2013. Students were approached during a practical class at the start of the semester, informed on the nature of the experiment and asked to participate in the two aspects of research. Firstly, they were asked to write the two tests, described in section 4.1.1. Secondly, they were asked to volunteer to play the game throughout the semester. Those who volunteered to play the game were required to write the two tests as a prerequisite for taking part in the research. The students were also informed that those who volunteered for and completed the game-play trails would receive compensation. Prerequisites set forth by the University of Stellenbosch's ethics board required several of the steps in this procedure to be administered by the class lecturer, which complicated communication throughout the experiment.

Two weeks after the initial test, students who indicated interest in taking part in the game-play experiment were contacted with instructions and a download link to a copy of the game. At this point, the version of the game sent to the students contained a limited level set with features fitting the workload they had done thus far.

One week later, a second version of the game containing the full level set was sent to the students. At this point, the students had covered the topics analogous to those covered in the game.

During the process of playing the game, students were asked to report on any technical issues. Aside from two students who did not have home computers with sufficient screen resolutions to support the game - an unforeseen occurrence given the game's relatively small resolution - no technical issues were reported by the players.

Finally, the students were approached during class for a second time a week before the end of the first semester. At this time, they were asked to write the second set of tests. In addition, the students who played the game were asked to complete the impression questionnaire detailing their experience in playing the game.

After this process the class lecturer gathered anonymised data of all four measurements, which in turn was used to determine the results presented in chapter 5.

4.4 Test Implementation Issues

While test implementation was executed without any issues, problems arose during the various tests for two reasons. Firstly, all but one of the measurements described required students to voluntarily complete sets of tests or surveys. Secondly, the sampled class was ninety-eight students at the start of the tests, meaning that any students not participating resulted in a significant reduction of responded.

Throughout the test-taking period there was a shortfall in the number of students who participated in any one test, meaning that only large discrepancies in the data gathered were discernible. These results are discussed in greater detail in Chapter 5.

The only test with a significant metric was the anonymised class marks, as this test provided a significant number of data points. This can be attributed to this result set not requiring direct student participation.

4.5 Summary

This chapter discussed the various measurements, tests and surveys used throughout the experiment to gather data.This included the concepts that prompted the design of these tests, the design process itself, as well as the sources of inspiration for these designs.

In addition, the sampling procedure used to determine how respondents was chosen was described. This included an explanation of the desired respondent and possible sources of such players within the local geographical environment.This chapter described the testing procedure and finally the issues that arose during this process were also detailed.

Following on the design and development decisions in Chapter 3 and 4, Chapter 5 describes the data gathering and analysis. This also highlights, if present, issues that arose in each of the tests. Finally, Chapter 6 concludes this thesis, giving the cumulative results of all the tests and recommending future paths of study.

Chapter 5

Measurement and Test Results

5.1 Introduction

This chapter analyses and discusses the data gathered as part of this research. As stated earlier, some of these sources contain too few data points to be useful in drawing any significant conclusions when considered in isolation. They are however included here for the sake of completeness as well as to inform any future researchers.

5.2 Language-Agnostic Test

As was discussed in Chapter 3, a programming language-agnostic test was developed to allow for comparison between the two test groups for which there was access to class marks for one but not the other. This was done so that the classroom mark comparison could underscore the results of the language-agnostic test and subsequently legitimise measurement of the groups without programming course marks, for who only the results of the language-agnostic test would be available.

Data was received in the form of an anonymised set of test scores with each entry representing the test scores of an individual respondent. While an initial 68 students completed the first test at the start of the semester, only 22 of those 68 also completed the second test. This is too low a number of respondents from which meaningful insight can be drawn, and was further compounded by the group being split into two - those who played the game and those who did not. These groups were sized at 45 and 28 respondants at the start of the semester and 18 and 4

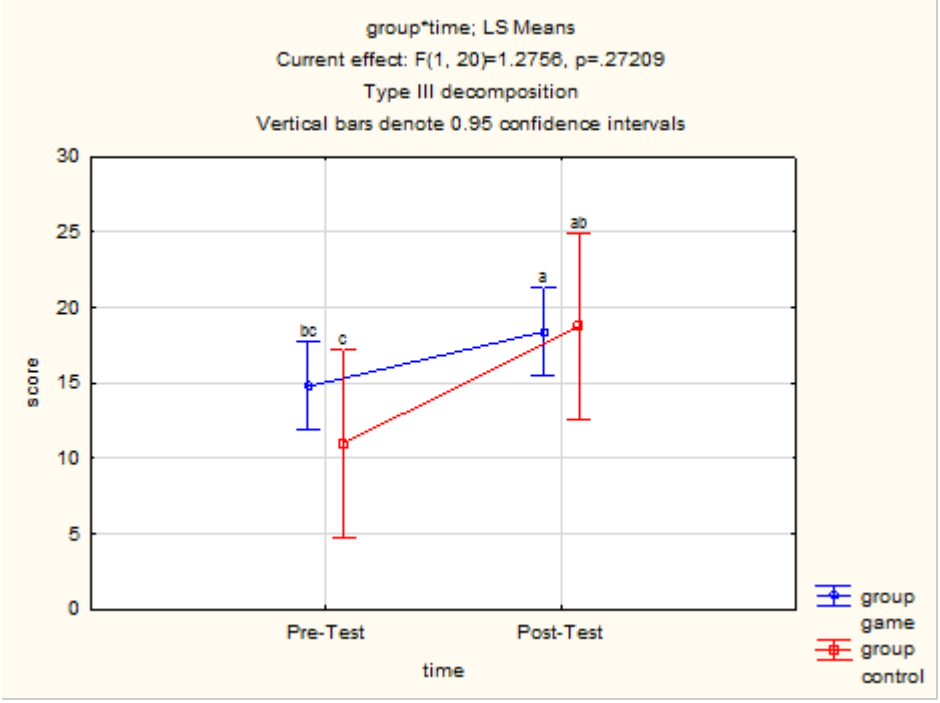

Figure 5.1: Mean comparison considering group and time differences.

respondants respectively, at the end of the semster.

The initial unequal group sizes were acceptable because of the longitudinal nature of the study and the expectation that the number of students who volunteer to write the tests and to play the game would certainly decrease as the semester progressed. Thus, although this would hinder statistical comparison because the sizes of the groups could differ drastically, it increased the likelihood of a larger number of final respondents.

Table 5.1: Fixed Effect Test for language-agnostic test results.

Effect	Num. DF	Den. Df	F	p
A	1	20	0.389594	0.539568
B	1	20	9611227	0.005642
A+B	1	20	1.275571	0.272089

*** A - Group Effect — B - Time — A+B - Group and Time effect*

Figure 5.1 shows the comparison between the test groups, split by the sampling

Table 5.2: The p-value of any group being distinct when compared to each other group.

	P+T1	P+T2	NP+T1	NP+T2
P+T1		0.031621	0.264972	0.241885
P+T2	0.031621		0.036357	0.913789
NP+T1	0.264972	0.036357		0.02989
NP+T2	0.241885	0.913789	0.02989	

*** P+T1 - Played game and did Test one — P+T2 - Played game and did Test two — NP+T1 - Control and did Test one — NP+T2 - Control and did Test two*

Table 5.3: Least Significant Difference (LSD) test between groups.

1st Mean	2nd Mean	Mean Diff.	Std. Error	p-value
P+T1	P+T2	-3.6111	1.562608	0.031621
P+T1	NP+T1	3.77778	3.293673	0.264927
P+T1	NP+T2	-3.97222	3.293673	0.241885
P+T2	NP+T1	7.38889	3.293673	0.036357
P+T2	NP+T2	-0.36111	3.293673	0.913789
NP+T1	NP+T2	-7.75000	3.314792	0.029890

*** P+T1 - Played game and did Test one — P+T2 - Played game and did Test two — NP+T1 - Control and did Test one — NP+T2 - Control and did Test two*

process as well as over time, referencing the two separate tests written by each group. The characters at the top of the four bar graphs denote whether any two groups can be considered statisically different significant at a level of 95%. If a character is shared between any two bars, the groups are not considered different at a p-value of 0.05.

From this it is determined that both groups scored significantly better in their second test than in their first test. However, the two groups did not vary significantly from each other for either of the tests.

As can be seen in Table 5.1, the only effect to carry a significant p-value, that is, one less than 0.05, is the time effect. The probability of the four groups being statistically similar is 27.21%, much to high too be reliable for reporting purposes.

Table 5.2 shows the p-values of each group when compared to every other group. The number in red shows the combination of groups that can be considered statisti-

Table 5.4: Descriptive statistics for language-agnostic test.

	N	Mean	Std. Dev.	Std. Error	-0.95%	+0.95%
Total	44	16.27273	6.24449	0.941392	14.3742	18.1713
P	36	16.58333	6.48680	1.081133	14.3885	18.7782
NP	8	14.87500	5.13914	1.816959	10.5786	19.1714
T1	22	14.09091	4.83941	1.031766	11.9452	16.2366
T2	22	18.4555	6.81544	1.453057	15.4327	21.4763
P+T1	18	14.77778	4.82098	1.139315	12.3804	17.1752
P+T2	18	18.38889	7.51578	1.771486	14.6514	22.4763
NP+T1	4	11.00000	4.08248	2.041241	4.5039	17.4961
NP+T2	4	19.75000	2.21736	1.108678	15.2217	22.2783

*** P - Played game — NP - Control — T1 - Did test one — T2 - Did test two — P+T1 - Played game and did Test one — P+T2 - Played game and did Test two — NP+T1 - Control and did Test one — NP+T2 - Control and did Test two*

cally distinct from one another. Each combination will be displayed twice because of the nature of the table. Game group test one and two differ significantly from one another. Similarly, the results of control group test one and two are distinct from one another. Lastly, the results of the first test of the control group and the second test of the game group were distinctly different. This last comparison, however, does not lead to any logical insight into the data, as neither the time and type variable of the two groups match.

5.3 Previous Computer Science and Mathematics Experience

As part of the first language agnostic test, all participants were asked to indicate what their previous experience with mathematics and computer science was. This information could not be tied to individual marks because of limitations in the capturing process and needing to adhere to ethical standards. However, it is still useful information on the experiment group as a whole.

As seen in Table 5.5, a few of the students indicated previous exposure to program-

ming. The eight students who indicted having "More than one year [programming] education" would be familiar with the concepts covered in the game. The same might be said, although with far less assuarance, of some of the students with only one year's experience or those who have some degree of self-education. Again, due to ethical regulation limitations it was impossible to tie these experienced individuals to course marks. However, considering the similarity in the standard deviation of the marks of the three groups, as seen in Table 5.9, there was no discrepancy in how the groups performed.

Table 5.6 shows group experience with mathematics. The overwhelming majority of respondents completed Matric mathematics, with only a few of the students indicating a higher level of mathematical competency. Accordingly, the level of mathematical skill should be consistent over all the groups, and should not affect the outcome of the study.

Table 5.5: Responses for "Please indicate your history with computer science."

Description	**Count**
None	14
One Year Education of Introductory Course	9
More than one year education	8
Some Self-aught	4
Self-Taught and consider yourself competent	2

Table 5.6: Responses for "Please indicate your history with mathematics."

Description	**Count**
University Mathematics	4
Matric Mathematics	30
Matric Mathematics Literacy	1
Minimal High School Mathematics	0
Other	1

Table 5.7: Initial data groupings received and their sizes

Group Description	Size
Played the game and completed both language agnostic tests	18
Played the game and completed only the first language agnostic test	9
Did not play the game, but completed both language agnostic tests	5
Did not play the game, but completed the first language agnostic test	13
Did not participate in the study in any way	27

5.4 University Course Marks

The target groups' final semester marks for their introductory programming class served as the main data source for this project. Testing also took place during this single semester. Because class marks measure performance at specific points in time, this test is cross-sectional. It would, in other words, provide data for only a single point in time, namely the end of the semester. While this does not allow tracking the progress of the students over time, it does allow comparison between the data of the game-play volunteers with the rest of the students in the introductory class.

All data was anonymised by the class lecturer because of limitations by the University's ethics board. Thus, no identifying information on any individual respondents was available. The data was received in five groups. Descriptions of these groups, as well as the number of data point in each group, is listed in Table 5.7.

This data was compressed into three groups not on the basis of number of tests written, but by whether students participated in the study. The first and second groups, those who played the game, were combined, as were third and fourth groups, those who did not. This resulted in three groups, as described in Table 5.8.

Table 5.9 shows the descriptive statistics of these three groups. As can be seen, the averages of the three groups differ by up to 7%. The standard deviation of the three groups, however, are similar and show a similar distribution of marks.

Table 5.10 shows the statistical significance of every of these three groups when compared to each other group. As can be seen, none of the comparisons are

Table 5.8: Compressed data groupings received and their sizes

Group Description	Size
P	23
NP	22
O	27

*** P - Played the game — NP - Did not play the game, but participated in the study — O - Did not participate in the study in any way*

Table 5.9: Descriptive statistics of university marks

	N	Mean	Std. Dev.	Std. Error	-0.95%	+0.95%
Total	72	67.83333	14.40168	1.697255	64.44910	71.21756
P	23	71.95652	13.15114	2.742203	66.26954	77.64350
NP	22	67.09091	15.09623	3.218527	60.39762	73.78420
O	27	64.92593	14.54945	2.800044	59.17035	70.68150

*** P - Played the game — NP - Did not play the game, but participated in the study — O - Did not participate in the study in any way*

Table 5.10: The p-value of any group being distinct when compared to each other group

	P	**NP**	**O**
P		0.257577	0.087454
NP	0.257577		0.599601
O	0.087454	0.599601	

*** P - Played the game — NP - Did not play the game, but participated in the study — O - Did not participate in the study in any way*

significant at a p-value of 0.5 or less. However, the difference of 7% between the students who played the game and the students who did not participate in the study is significant at a p-value of 0.087. While this metric could improve it is nonetheless promising when considering the low number of respondents in each of the three groups.

5.5 Impression Survey

As part of the second language-agnostic test the students were asked to complete an impression survey, giving their opinion on the game. This was presented to the students as an optional survey they could complete after completing the second language-agnostic test. This meant that only twelve of the respondents who played the game completed the impression survey. The impression survey consisted of two quantitative questions to determine the level of respondent engagement with the game. Following these were twelve 5-point Likert scale questions to qualitatively determine respondants' impressions of their experience with the game. The results of the impression survey are shown in Table 5.11

As seen from Table 5.11, respondants who completed the survey were evenly spread in the percentage of the game they completed, though there is a slightly higher concentration of respondents at the lower end of the distribution.

Some inferences can be made when looking at the impression questions. Due to the small number of respondents, these inferences are subjectively based on the limited evidence present, and further testing is required to make more definite claims.

Firstly, the majority of the respondents had generally positive feedback regarding their experience with the game. Secondly, respondents agreed that the mechanics introduced in the game helped them to understand and make decisions with regarding in-game puzzles. Most importantly questions dealing with the effect of the game on programming skill were responded to positively, suggesting that users feel as though more easily cope with programming problems after exposure to the game developed for this research.

Table 5.11: Impression Survey: number of respondents per answer category

	<1h	**1-2h**	**2-4h**	**4-6h**	**>6h**
How much time would you estimate you spent playing the game?	5	4	0	2	1
	<20%	**20-40%**	**40-60%**	**60-80%**	**>80%**
How many of the level sets would you estimate you completed?	3	3	3	0	3
Please indicate if you agree or disagree with each of the following statements.	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
I found the game to be easy overall.	1	4	6	1	0
I could complete at least the first few levels of each level set.	7	4	1	0	0
I had difficulty following and completing the tutorial level sets.	1	3	3	1	4
I accurately could predict what if a solution to a level was going to work before actually running it.	2	3	6	1	0
I had trouble making placements when there were a lot of symbols on the screen.	0	4	4	3	1
Watching the carrier as it moved helped me realise where I had made a mistake.	3	7	2	0	0
Being able to see the size and colour of a gem at all times helped me understand what was going on in my solutions.	2	7	3	0	0
Knowing the size and colour of a gem at a specific point helped me make decisions about how to approach a puzzle.	1	8	3	0	0
Seeing the size and colour of a gem change when a level ran helped me to better understand the function of specific symbols.	2	8	2	0	0
Playing the game helped me to more easily derive solutions for puzzles.	1	6	5	0	0
Playing the game gave me insight into the way programming languages work.	1	5	4	1	1
I'll find it easier to solve programming problems after playing the game.	0	6	4	1	1

5.6 Summary

This chapter discussed the results of tests, surveys, and analyses done to answer the research questions as presented in Chapter 1. The results of the language-agnostic test developed as part of this research were shown. This chapter considered the previous programming experience of the respondents, as well as their impressions of the game played. Finally, it examined the university course marks of the various groups and discussed these differences amongst the groups.

Emphasised in the discussion of all the results is the impact that a lack of respondents had on the measurable metrics. Given that the research is longitudinal and requires a long term commitment from the respondents, a decline in these respondents was expected. This, coupled with the inherently limited scope of this research, meant that the lack of respondents led to ome of the results being inconclusive. However, taken as a whole, the results gathered seem reliable enough to draw some conclusions, which will be more fully discussed in Chapter 6.

Chapter 6

Conclusion

The research presented in the preceding chapters explored design, development, and implementation of a game aimed at conveying introductory programming concepts. This chapter concludes the thesis with a holistic overview of the results gathered, their impact given the hypotheses, as well as recommendations for future research.

6.1 Objectives

As stated in Chapter 1, the aim of the research is to show the use of computer games as a medium for conveying the core principles of computer programming. This required the completion of two objectives:

Firstly, it required the design and development of a game aimed at conveying the introductory principles of computer science. This game had several requirements:

1. The game must convey a coherent set of the basic programming concepts.
2. The game must be understandable to users without any outside input.
3. The game must be free of any technical issues which would have cause it to crash.
4. The game may not contain any conceptual fallacies that could potentially corrupt the concepts it must convey.
5. The game must run on a relatively wide array of computers, so that its target audience can easily access it.

Secondly, the research required testing the game to measure its success. This section of the research was divided into the following subsections:

1. The location of a group of respondents on which to test the effectiveness of the game as a tool for conveying the above mentioned concepts.
2. The identification and gathering of any relevant data on the respondents.
3. If necessary, the development of a measurable testing mechanism that can be given to the respondents.
4. The administration of any the testing process.
5. The processing and analysis of any gathered data.

6.2 Hypotheses

Given the two research questions, there are possible hypotheses and one possible null hypothesis.

- H0: No correlation was found between exposure to an electronic serious game and understanding of introductory concepts.
- H1: A correlation was found between exposure to an electronic serious game and understanding of introductory concepts, given the additional presence of a standard introductory programming course.
- H2: A correlation was found between exposure to an electronic serious game and understanding of introductory concepts, given no other directly relevant stimuli.

6.3 Summary of Results

Chapter 5 discussed the results of the various measurement mechanics used during this research. Due to a lack of volunteers on some of the tests, or a decline in volunteers over the course of the experiment, many of the results gathered were not as definitive as desired. Given the scope of the research,as well as the limitations imposed by qualitative, respondent based research, the lack of respondents is not entirely surprising.

However, the gathered data suggests positive support for the primary hypothesis, namely, that electronic games can be used to enhance established introductory programming courses. It is important to note that, due to the nature of respondent-based social research, it is impossible to definitively state that no other factors influenced these results. The primary metrics of this research, the university course marks, resulted in a difference of approximately 7% between respondents who played the game, and those who did not participate in the study. This is significant at a p-value of 0.1. Given the relatively small size of the sample group, these results lend themselves to the prospect of future research aimed at clarification.

While the rest of the measurements are not as definitive because of their qualitative nature, the limited number of respondents, or a combination of the two factors, they suggest, overall, that exposure to the game had a positive effect on both the academic outcomes of the respondents and their outlook on their programming skills development.

It should be noted that these outcomes can be explained by other factors, such as a volunteer bias in the respondents or a positivity bias in the surveys. While these possibilities should be explored, such exploration would have been impossible given the scope of this research as it would have further diluted the small sample pool and made significant result sets extremely unlikely.

This research specifically dealt with the creation of a codeless, symbol-based learning environment and, in combination with the work done by others in this field as described in Chapter 2 (Marques et al., 2012; Kelleher et al., 2007), there is a clear pattern indicating the positive influence of games in basic programming education. These studies did not use directly comparable metrics, but all of them do show a positive effect in programming education amongst learners exposed to programming serious games.

6.4 Comparison with Previous Approaches

As was described in Chapter 3, the game developed here made the novel leap of representing programming logic through visual symbols rather than a more traditional text based representation. It is therefore pertinent to compare the results from this research to some of the previous approaches to teaching programming fundamentals through serious games.

Unfortunately there is no standard method of data gathering or modeling across the previous work in this field, which makes any direct comparison impossible. In addition, since most of the work has dealt with direct respondents in some

way various known and unknown situational factors would make any definitive conclusion from comparison impossible. It is therefore more prudent to consider the essence of the results from the various pieces of work and see if an overall trend is indicated.

In 2006, Kelleher compared two version of an internal tool, Alice, to measure the effect narratives have on user engagement and found a positive correlation between the use of narratives in games and user engagement (Kelleher et al., 2007).

Marques, in 2012, used the time taken to complete their games levels as a metric for comparison (Marques et al., 2012). Additionally, similar to the research presented here, they studied the effect on university students and reported an average increase of 10% amongst the 14 students whose university course marks they compared. Lastly, they gathered qualitative feedback indicating that the game is rewarding and fun.

These two approaches indicate that games have the ability to engage users, while also having the ability to increase performance. This aligns closely with the findings of this research. While neither of these two pieces of previous work, nor the work presented here, can be taken as conclusive on their own or as a whole, it does seem like they indicate the same optimistic trend.

Unfortunately it does not seem that the use of symbols over text as a representation method made much difference in the effectiveness of the approach. However, it does not seem to have performed significantly worse either, at least when compared with Marques. This might indicate that a symbol based approach might be of significant interest in situations where language is a barrier to adoption, such as with the very young of the illiterate.

6.5 Future Work

6.5.1 Restrictions and Limitations

Respondent Drop Off and Subsequent Scope Limitations

The research plan was to use the first semester of 2013 to gather data from a introductory programming class at the Department of Information Sciences at Stellenbosch University. The second semester would be used to gather data from a second homogeneous source to test whether games are successful as replacement educational tools as opposed to only as enhancement.

Initially, the response rate in the class was positive, with almost two-thirds of the students participating in the voluntary skills assessment test developed as part of the research. In addition, just under half the class volunteered to play the game. Although this resulted in an uneven distribution of students between the three groups of students who a) did not participate, b) only wrote the test, and c) played the game, it was suspected that many of the students in groups b and c would not fulfil their commitment as research subjects for the duration of the experimen thus, it was decided to maximise the number of volunteers to have potentially as large a sample at the end of the semester as possible.

At the end of the first semester, only twenty-three students had finished the game. The majority of the respondents who completed the aptitude tests at both the start and the end of the semester were also part of the group who played the games. This meant that the number of results in the control group were to few to provide statistically significant results. However, as the class marks were available irrespective of respondent interaction, the results gathered from that data was statistically significant.

This, however, meant that any data gathered from the second series of tests would not be useful in proving the research hypothesis as there was no control group with which it could be compared. This was because the second test set would only be supported only by data from the aptitude test.

As a result, the second set of tests was abandoned and instead the focus fell solely on class marks as the measure for determining the validity of the main hypothesis.

Limited Sample Selection

As has been mentioned, this research used student volunteers as respondents. Although a limited form of financial incentive was provided, the research did not officially partner with any department and could not require students to participate as part of their course. This meant that there was limited sampling control in the experiment, which made it impossible to spread out of factors such as previous informal programming experience.

Groups would ideally be used to isolate not only planned influences, but also other variables, such as volunteer bias.

6.5.2 Next Steps

Separation of Influences through Larger Sample Groups

At the time of writing, several studies have been done that explore the possibility of using electronic games to teach programming. This has resulted in the development of several prototype tools that have shown various degrees of success in their respective studies. However, as has been previously highlighted, direct comparison between the gathered data of different studies is impossible because the testing mechanism used in this study and those developed by previous studies are unique. These testing mechanisms differ in various ways. Firstly, previous studies focused on different population samples, with some focusing on young children and others focusing on adults. Secondly, these studies differed in terms of their explicit research goals. Lastly, they differ in the type of data the tests generate.

When considered as a whole, separate data sets that all point to the same conclusion that electronic games can be used as enhancement tools for programming courses. In addition, several distinct and different tools have been developed.

This creates a ripe environment for a meta analysis of the previous approaches, with special focus given to testing these solutions in such a manner that they produce comparable data. This is not an easy task to accomplish. Such an experiment would require a large number of respondents to make each test group large enough for data comparison.

The following groups would be required for the suggested experiment: three groups testing the effect of three distinct games in conjunction with a class environment; three groups testing the same three games in isolation from a class environment; two groups, to test a game not expected to affect programming skills, with one group in a class environment and one in an isolated environment to control for a volunteer or participation bias; one group to test the effects of a class without a game influence; and, finally, one group to test only the effect of the tests over time, without the influence of either a class or a game.

This makes for a total of twelve distinct groups, needing a large number of homogeneous respondents to test the distinctions between the data gathered in such an experiment.

Qualitative Prototype Testing

One of the objectives implicit in this research was the development of a prototype fit for testing the hypotheses highlighted in Chapter 1. One of the requirements of this objective is that the prototype must be programmatically stable. It should not crash and should run on all targeted devices to reach its intended audience.

One of the instructions the respondents of this research received was to report any technical issues. A couple of respondents reported issues with screen resolutions, but otherwise no complaints were received. While this can be deemed as succeeding in the technical development of the task, it is recommended that any future research in to this field includes questioning respondents specifically about their experience with the stability and usability aspects of the prototype.

This would allow future researcher to more easily identify the potential influence of any technical issues with the results of those respondents. In the research presented here, any data from users experiencing technical issues was simply ignored in order to avoid fragmenting the number of influences in a group.

6.5.3 Incorporating fields in Visualisation

Section 3.2.2 briefly mentioned a number of fields that have benefited from studies in visualisation. Incorporating elements from these fields was outside of the scope of this study, but such integration would likely have significant positive impacts on the ability of the game to reach it's targets.

A future prototype specifically aimed at incorporating lessons learned from these fields into order to increase usability, concept conveyance as well as potentially teaching new lessons could be an interesting field of future study.

Bibliography

Abt, C. C. (1970). Serious games: The art and science of games that simulate life. *Viking Compass Book, USA*.

America's Army Home Page (2013). [Online] Available: http://www.americasarmy.com/ [24 October 2015].

BotLogic Home Page (2013). [Online] Available: http://botlogic.us/ [25 October 2015].

BYOB Home Page (2012). [Online] Available: http://snap.berkeley.edu/ [16 October 2015].

Carmigniani, J., Furht, B., Anisetti, M., Ceravolo, P., Damiani, E., & Ivkovic, M. (2011). Augmented reality technologies, systems and applications. *Multimedia Tools and Applications*, *51*(1), 341–377.

Caudwell, A. (2009). Gource: Software version control vizualisation. [Online] Available: https://code.google.com/archive/p/gource/[25 October 2015].

Code Hero Project Page (2012). [Online] Available: http://primerlabs.com/codehero [23 October 2015].

Codecademy Home Page (2012). [Online] Available: http://www.codecademy.com [25 October 2015].

CodeSpells Game Page (2013). [Online] Available: http://codespells.org/ [16 October 2015].

Conti, J. (2006). The Internet of Things. *Communications Engineer*, *4*(6), 20–25.

Crawford, C. (2002). *Art of Interactive Design*. No Starch Press San Francisco, CA.

Crenshaw, T., Chambers, E., & Metcalf, H. (2008). A case study of retention practices at the University of Illinois at Urbana-Champaign. *ACM SIGCSE Bulletin*, *40*(1), 412–416.

Currie, E. (2006). *Fundamentals of Programming using Java*. Thomson Learning.

Dehnadi, S. (2006). Testing programming aptitude. In *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, (pp. 22–37).

Derryberry, A. (2007). Serious games: online games for learning. *Adobe Whitepaper, November*.

else{Heart.break()} Home Page (2012). [Online] Available: http://eriksvedang.com/2012/02/16/else-heart-break/ [23 October 2015].

Entis, G. (2011). Beyond Badges – Gamification for the Real World. *Entertainment Computing–ICEC 2011*, (pp. 472–472).

Farrel, J. (2013). *A beginner's guide to Programming Logic and Design: Comprehensive*. Cengage Learning.

Farrell, J. (2012). *Microsoft Visual C# 2012 An Introduction to Object-Orientated Programming*. Cengage Learning.

Fitocracy Home Page (2012). [Online] Available: http://www.galaxyzoo.org/ [25 October 2015].

Foldit Science Page (2013). [Online] Available: http://fold.it/portal/info/science [23 October 2015].

Galaxy Zoo Home Page (2012). [Online] Available: https://www.fitocracy.com/ [25 October 2015].

Google Blocky Home Page (2013). [Online] Available: https://blockly-games.appspot.com/ [25 October 2015].

Harwani, B. (2012). *Introduction to Python Programming and Developing GUI Applications with PyQT*. Cengage Learning.

Hayden, D. (2011). Gamification by Design – Implementing Game Mechanics in Web and Mobile Apps. [Online] Available: http://www.davidhayden.me/blog/gamification-by-design-implementing-game-mechanics-in-web-and-mobile-apps [24 October 2015].

Ingress Home Page (2013). [Online] Available: http://www.ingress.com/ [25 October 2015].

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 1455–1464). ACM.

Khatib, F., Cooper, S., Tyka, M. D., Xu, K., Makedon, I., Popović, Z., Baker, D., & Players, F. (2011). Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences*, *108*(47), 18949–18953.

Lambert, K. (2011). *Fundamentals of Python: From first programs through data structures*. Cengage Learning.

Logo Foundation Home Page (2013). [Online] Available: http://el.media.mit.edu/logo-foundation/ [25 October 2015].

Lucidchart - Activity Diagram Introduction Page (2015). [Online] Available: https://www.lucidchart.com/pages/uml/activity-diagram [27 October 2015].

Manufactoria Game Page (2013). [Online] Available: http://www.pleasingfungus.com/Manufactoria/ [25 October 2015].

Marques, B. R., Levitt, S. P., & Nixon, K. J. (2012). Video games as a medium for software education. In *Games Innovation Conference (IGIC), 2012 IEEE International*, (pp. 1–4). IEEE.

Mughal, K., Hamre, T., & Rasmussen, R. (2008). *Java Actually: A comprehensive primer in programming*. Cengage Learning.

Muratet, M., Torguet, P., Viallet, F., & Jessel, J. (2010). Experimental Feedback on Prog&Play: A Serious Game for Programming Practice. In *Computer Graphics Forum*. Wiley Online Library.

Muwanga-Zake, J. (2003). Is science education in a crisis? some of the problems in South Africa. [Online] Available: http://www.scienceinafrica.co.za/scicrisis.html [04 March 2012].

Quest to Learn Home Page (2012). [Online] Available: http://q2l.org/ [25 October 2015].

Raskin, J. (2000). *The humane interface: new directions for designing interactive systems*. Addison-Wesley Professional.

Ribbon Hero 2 Home Page (2013). [Online] Available: http://www.ribbonhero.com/news.html [25 October 2015].

Scratch Home Page (2012). [Online] Available: http://scratch.mit.edu/ [25 October 2015].

Small Business Labs (2012). [Online] Available: http://www.smallbizlabs.com/2011/02/what-is-gamification.html [24 October 2015].

Smith, J. (2011). The growing internet of things. [Online] Avalable: http://www.capacitymagazine.com/Article/2919323/The-growing-Internet-of-Things.html [04 March 2012].

Stack Overflow User Page (2014). [Online] Available: http://https://stackoverflow.com/users [5 June 2014].

Stack Overflow Website (2013). [Online] Available: http://stackoverflow.com [25 October 2015].

Tufte, E. R., & Graves-Morris, P. (1983). *The visual display of quantitative information*, vol. 2. Graphics press Cheshire, CT.

Unger, R., & Moult, J. (1993). Genetic algorithms for protein folding simulations. *Journal of molecular biology*, *231*(1), 75–81.

Unity Home Page (2012). [Online] Available: http://unity3d.com/unity/ [23 October 2015].

Yaroslavski, D. (2012). Lightbot 2 Game Page. [Online] Available: http://armorgames.com/play/6061/light-bot-20 [23 October 2015].

Yo Yo Games (2012). GameMaker Home Page. [Online] Available: http://www.yoyogames.com/make [23 October 2015].

Zelle, J. (2010). *Python Programming: An Introduction to computer science 2nd edition*. Franklin, Beedle and Associates.

Zicherman, G. (2010). Fun is the future: Mastering Gamificaton. [Online] Available: http://www.youtube.com/watch?v=6O1gNVeaE4g [25 October 2015].

Appendix A

First language agnostic programming test

This appendix gives, verbatim, the first aptitude test presented to the respondents at the start of the testing period.

The aptitude test is split into a preamble and 5 sections, the purpose of which are described below.

The preamble is aimed at determining the respondent's existing knowledge of programming and mathematics, which could provide data about the respondent's predisposition to answering the test easily.

Section A aims to introduce the respondent to the format of the test. It introduces them to the idea that the test makes use of sequential statements of simple mathematical arithmetic, and that they will be asked to track the values of various variables.

Section B introduces each type of question the respondents will be asked to answer. Each question is introduced with an example, followed by an initial question of the same difficulty level. Each type of question is roughly related to a segment of programming knowledge.

Question B1 introduces conditional statements.

Question B2 introduces bounded loops.

Question B3 introduces loops with conditional escapes.

Question B4 introduces programming functions.

Questions B5 introduces memory pointers.

Sections C, D and E then iterate on these questions in increasing, sequential, difficulty.

In addition to these explicit differentiation, the test as a whole also introduces basic boolean logical operators to the respondent. Respondents are exposed to AND, OR and NOT operators by the end of the test, as those are all logically coherent in plain English.

Please indicate you history with computer science:

	None
	One Year Education or Introductory Course
	More than One Year Education
	Some Self-Taught
	Self-Taught and considered competent

Please indicate your history with Mathematics

	University Math
	Matric Math
	Minimal High school Math
	Other

Section A

Example:

Given that:

x has a starting value of 0
y has a starting value of 10
z has a starting value of 6

And the following commands:

```
x ← y
y ← 2 × x
z ← y + x
```

with the resulting values:

x has an end value of 10
y has an end value of 20
z has an end value of 30

Question 1

Given That

a has a starting value of 7
c has a starting value of 19

And the following commands:

```
a ← a + 3
c ← c + a - 4
a ← c − a
```

what are the end values of a and c?

Question 2

Given:

a has a starting value of 5
c has a starting value of 10

And the following commands:

```
a ← c ÷ a
b ← c
c ← b ÷ a
d ← a + b - c
```

What are the end values of a,b,c and d?

Question 3

Given:

a has a starting value of 5
c has a starting value of 10

And the following commands:

```
b ← a ÷ c
c ← a × b
d ← a + c × b
```

What are the end values of a,b,c and d?

Section B

Example 1:

Given:

a has a starting value of 10

And the following commands:

```
if a is 15 then do:
     b ← 5
otherwise do:
     b ← 7
```

then a will have an end value of 10
and b will have and end value of 7

Question 1

Given:

a has a starting value of 2
b has a starting value of 3

And the following commands:

```
if (a+b) is 5 then do:
     a ← a - 1
     b ← b - 1
otherwise do:
     a ← a + 1
     b ← b + 1
```

What are the end values of a and b?

Example 2:

Given:

a has a starting value of 1

And the following commands:

repeat(10)
    a ← a + 1

then a will have an end value of 11

Question 2

Given:

a has a starting value of 1

And the following commands:

```
repeat(4)
    a ← a×2
```

What is the end value of a?

Example 3:

Given:

a has a starting value of 1
b has a starting value of 5

And the following commands:

while a is not 5 do:
    a ← a + 1
    b ← b + 1

then a will have an end value of 5
and b will have an end value of 9

Question 3

Given:

a has a starting value of 1
b has a starting value of 1

And the following commands:

```
while a is not 5 do:
      a ← a + 1
      b ← b×a
```

What is the end value of a and b?

Example 4:

Given:

a has a starting value of 1

And the following commands:

```
to add_five with var do:
        var ← var + 5

add_five(a)
add_five(a)
```

then a will have an end value of 11

Question 4

Given:

a has a starting value of 10
b has a starting value of 22

And the following commands:

```
to double with var do:
      var ← var+var

double(a)
double(b)
```

What are the end values of a and b?

Example 5:

Given:

a has a starting value of 10

And the following commands:

b <<< a

a ← a + 10

then a will have an end value of 20
and b* will have an end value of 20

Question 5

Given:

a has a starting value of 5
b has a starting value of 12

And the following commands:

```
c <<< a + b
a ← a + 10
b ← b + 8
```

What is the end values of a, b and c*

Section C

Question 1

Given:

a has a starting value of 10

And the following commands:

```
if a is 10 then do:
     if a is not divisible by 5 then do:
          b ← 1
     otherwise do:
          b ← 2
otherwise do:
     if a is divisible by 3 then do:
          b ← 3
     otherwise do:
          b ← 4
```

What is the end value of b?

Question 2

Given:

a has a starting value of 1
b has a starting value of 1

And the following commands:

```
repeat(2)
     repeat(2)
          repeat(2)
               a ← a + b
```

What is the end value of a?

Question 3

Given:

a has a starting value of 1
b has a starting value of 1

And the following commands:

```
while a is not 5 do:
     a ← a + 1
     while (a+b) is a multiple of 2
          b ← b + 1
```

What are the end values of a and b?

Question 4

Given:

a has starting value of 7

And the following commands:

```
to split with var do:
     var ← var ÷ 2

to increase with var do:
     var ← var + 1

increase(a)
split(a)
a ← a + 2
```

What are the end values of a?

Question 5

Given:

a has a starting value of 2

b has a starting value of 8

And the following commands:

```
c <<< a

d <<< b + a

b ← b + a
```

What are the end values of a, b, c* and d*?

Section D

Question 1

Given:

a has a starting value of 10

And the following commands:

```
if a is not 10 then do:
     a ← a + 6
     if a is divisible by 4 then do:
          b ← 1
     otherwise do:
          b ← 2
otherwise do:
     a ← a - 4
     if a is divisible by 3 then do:
          b ← 3
     otherwise do:
          b ← 4
```

What is the end value b?

Question 2

Given:

a has a starting value of 1
b has a starting value of 1

And the following commands:

```
repeat(2)
     repeat(2)
          repeat(2)
               a ← a + b
     b ← b + 1
```

What is the end value of a and b?

Question 3

Given:

a has a starting value of 1
b has a starting value of 1

And the following commands:

```
while a is smaller than 20 do:
     a ← a + b
     while (a+b) is a multiple of 2 do:
          b ← b + a
```

What is the end value of a and b?

Question 4

Given:

a has a starting value of 5
b has a starting value of 3

And the following commands:

```
to mul with base, deg do:
     base ← base*deg

mul(a,b)
```

What are the end values of a and b?

Question 5

Given:

a has a starting value of 7

b has a starting value of 9

And the following commands:

```
c <<< b

b ← b + a

d <<< a

a ← a + b

e <<< a - b
```

What are the values of a, b, c*, d* and e*?

Section E

Question 1

Given:

a has a starting value of 10
b has a starting value of 8

And the following commands:

```
if a is divisible by 2 or b is divisible by 2 do:
     c ← 1
     if a is divisible by 4 and b is divisible by 4 do:
          d ← 1
     otherwise do:
          d ← 2
otherwise do:
     c ← 2
     if a is divisible by 4 and b is divisible by 4 do:
          d ← 3
     otherwise do:
          d ← 4
```

What are the end values of c and d?

Question 2

Given:

a has a starting value of 13
b has a starting value of 12

And the following commands:

```
if (a + b) is divisible by 2 do:
     a ← a + 2
if (a + b) is divisible 5 do:
     b ← b + 10
while b is smaller than 20 do:
     a ← a + 1
     b ← b + 1
c ← 1
while a is smaller than 15 do:
     a ← a + 1
     c ← c + 1
```

What are the end values of a, b and c?

Question 3

Given:

a has a starting value of 2
b has a starting value of 10

And the following commands:

```
to incr with base, deg do:
     base ← base+deg

while a is smaller than 20 do:
     if (a+b) is divisible by 2 do:
          incr(a,b)
          a ← a + 1
     otherwise do:
          incr(b,a)
```

What are the end values of a and b?

Appendix B

Second language agnostic programming test

This appendix gives, verbatim, the second aptitude test presented to the respondents at the end of the testing period.

This almost identical to the initial test presented in appendix A, with different variable values. This protects against respondents copying the answers from their initial tests.

Please indicate you history with computer science:

	None
	One Year Education or Introductory Course
	More than One Year Education
	Some Self-Taught
	Self-Taught and considered competent

Please indicate your history with Mathematics

	University Math
	Matric Math
	Minimal High school Math
	Other

Section A

Example:

Given that:

x has a starting value of 0
y has a starting value of 10
z has a starting value of 6

And the following commands:

```
x ← y
y ← 2 × x
z ← y + x
```

with the resulting values:

x has an end value of 10
y has an end value of 20
z has an end value of 30

Question 1

Given That

a has a starting value of 5
c has a starting value of 22

And the following commands:

```
a ← a + 3
c ← c + a - 4
a ← c − a
```

what are the end values of a and c?

Question 2

Given:

a has a starting value of 10
c has a starting value of 5

And the following commands:

```
a ← c ÷ a
b ← c
c ← b ÷ a
d ← a + b - c
```

What are the end values of a,b,c and d?

Question 3

Given:

a has a starting value of 14
c has a starting value of 7

And the following commands:

```
b ← a ÷ c
c ← a × b
d ← a + c × b
```

What are the end values of a,b,c and d?

Section B

Example 1:

Given:

a has a starting value of 10

And the following commands:

```
if a is 15 then do:
     b ← 5
otherwise do:
     b ← 7
```

then a will have an end value of 10
and b will have and end value of 7

Question 1

Given:

a has a starting value of 3
b has a starting value of 4

And the following commands:

```
if (a+b) is 5 then do:
     a ← a - 1
     b ← b - 1
otherwise do:
     a ← a + 1
     b ← b + 1
```

What are the end values of a and b?

Example 2:

Given:

a has a starting value of 1

And the following commands:

repeat(10)
    a ← a + 1

then a will have an end value of 11

Question 2

Given:

a has a starting value of 1

And the following commands:

```
repeat(3)
    a ← a+a+2
```

What is the end value of a?

Example 3:

Given:

a has a starting value of 1
b has a starting value of 5

And the following commands:

while a is not 5 do:
    a ← a + 1
    b ← b + 1

then a will have an end value of 5
and b will have an end value of 9

Question 3

Given:

a has a starting value of 3
b has a starting value of 1

And the following commands:

```
while a is less than 5 do:
      a ← a + 1
      b ← b×a
```

What is the end value of a and b?

Example 4:

Given:

a has a starting value of 1

And the following commands:

```
to add_five with var do:
      var ← var + 5

add_five(a)
add_five(a)
```

then a will have an end value of 11

Question 4

Given:

a has a starting value of 5
b has a starting value of 11

And the following commands:

```
to triple with var do:
      var ← var × 3

triple(a)
triple(b)
```

What are the end values of a and b?

Example 5:

Given:

a has a starting value of 10

And the following commands:

b <<< a

a ← a + 10

then a will have an end value of 20
and b* will have an end value of 20

Question 5

Given:

a has a starting value of 3
b has a starting value of 9

And the following commands:

```
c <<< a + b
a ← a + 10
b ← b + 8
```

What is the end values of a, b and c*

Section C

Question 1

Given:

a has a starting value of 10

And the following commands:

```
if a is 10 then do:
     if a is not divisible by 5 then do:
          b ← 1
     otherwise do:
          b ← 2
otherwise do:
     if a is divisible by 3 then do:
          b ← 3
     otherwise do:
          b ← 4
```

What is the end value of b?

Question 2

Given:

a has a starting value of 2
b has a starting value of 1

And the following commands:

```
repeat(2)
     repeat(2)
          repeat(3)
               a ← a + b
```

What is the end value of a?

Question 3

Given:

a has a starting value of 1
b has a starting value of 2

And the following commands:

```
while a is less than 5 do:
     a ← a + 1
     while (a+b) is a multiple of 2
          b ← b + 1
```

What are the end values of a and b?

Question 4

Given:

a has starting value of 9

And the following commands:

```
to split with var do:
     var ← var ÷ 2

to increase with var do:
     var ← var + 1

increase(a)
split(a)
a ← a + 2
```

What are the end values of a?

Question 5

Given:

a has a starting value of 8

b has a starting value of 2

And the following commands:

```
c <<< a

d <<< b + a

b ← b + a
```

What are the end values of a, b, c* and d*?

Section D

Question 1

Given:

a has a starting value of 9

And the following commands:

```
if a is not 10 then do:
     a ← a + 6
     if a is divisible by 4 then do:
          b ← 1
     otherwise do:
          b ← 2
otherwise do:
     a ← a - 4
     if a is divisible by 3 then do:
          b ← 3
     otherwise do:
          b ← 4
```

What is the end value b?

Question 2

Given:

a has a starting value of 2
b has a starting value of 2

And the following commands:

```
repeat(2)
     repeat(2)
          repeat(3)
               a ← a + b
     b ← b + 1
```

What is the end value of a and b?

Question 3

Given:

a has a starting value of 1
b has a starting value of 3

And the following commands:

```
while a is smaller than 20 do:
     a ← a + b
     while (a+b) is a multiple of 2 do:
          b ← b + a
```

What is the end value of a and b?

Question 4

Given:

a has a starting value of 5
b has a starting value of 3

And the following commands:

```
to mul with base, deg do:
     base ← base*deg

mul(a,b)
```

What are the end values of a and b?

Question 5

Given:

a has a starting value of 9

b has a starting value of 7

And the following commands:

```
c <<< b

b ← b + a        b = 16

d <<< a

a ← a + b        a = 25

e <<< a - b
```

What are the values of a, b, c*, d* and e*?

Section E

Question 1

Given:

a has a starting value of 9
b has a starting value of 15

And the following commands:

```
if a is divisible by 2 or b is divisible by 2 do:
     c ← 1
     if a is divisible by 4 and b is divisible by 4 do:
          d ← 1
     otherwise do:
          d ← 2
otherwise do:
     c ← 2
     if a is divisible by 3 and b is divisible by 3 do:
          d ← 3
     otherwise do:
          d ← 4
```

What are the end values of c and d?

Question 2

Given:

a has a starting value of 11
b has a starting value of 9

And the following commands:

```
if (a + b) is divisible by 2 do:
     a ← a + 2
if (a + b) is divisible 5 do:
     b ← b + 10
while b is smaller than 20 do:
     a ← a + 1
     b ← b + 1
c ← 1
while a is smaller than 15 do:
     a ← a + 1
     c ← c + 1
```

What are the end values of a, b and c?

Question 3

Given:

a has a starting value of 10
b has a starting value of 2

And the following commands:

```
to incr with base, deg do:
     base ← base+deg

while a is smaller than 20 do:
     if (a+b) is divisible by 2 do:
          incr(a,b)
          a ← a + 1
     otherwise do:
          incr(b,a)
```

What are the end values of a and b?

Appendix C

Game impression survey

This appendix gives the impression survey that was presented to the respondents to optionally complete after they had played the game and done the two language agnostic tests.

How much time would you estimate you spent playing the game?

<1h 1-2h 2-4h 4-6h >6h

What many of the level sets would you estimate you completed?

<20% 20-40% 40-60% 60-80% 80-100%

All of the questions below need to be 5-point Likert-scale questions. ie with 5 answers: Strongly Disagree – Disagree – Neutral – Agree – Strongly Agree

I found the game to be easy overall

I could complete at least the first few levels of each level set

I had difficulty following and completing the tutorial level sets.

I accuratly could predict what if a solution to a level was going to work before actually running it.

I had trouble making placements when there were a lot of symbols on the screen.

Watching the carrier as it moved helped me realise where I had made a mistake.

Being able to see the size and colour of a gem at all times helped me understand what was going on in my solutions.

Knowing the size and colour of a gem at a specific point helped me make decisions about how to approach a puzzle.

Seeing the size and colour of a gem change when a level ran helped me to better understand the function of specific symbols.

Playing the game helped me to more easily derive solutions for puzzles.

Playing the game gave me insight into the way programming languages work

I'll find it easier to solve programming problems after playing the game.

Appendix D

Anonymised end-of-semester university marks

Both Tests and Played	**Both Tests, but did not Play**	**Test One, Not Two and Played**	**Test One, Not Two and Did Not Play**	**Did Not Participate**	**Played Game**	**Did not play game**	**Did Not Participate**
45	45	63	45	**40**	45	45	**40**
50	55	67	50	**45**	50	55	**45**
58	58	67	51	**50**	58	58	**50**
59	58	68	57	**50**	59	58	**50**
63	67	76	60	**52**	63	67	**52**
64	75		61	**53**	64	75	**53**
67	76		61	**54**	67	76	**54**
68	84		61	**56**	68	84	**56**
75	90		75	**57**	75	90	**57**
78			82	**57**	78	45	**57**
78			83	**58**	78	50	**58**
80			86	**59**	80	51	**59**
80			96	**59**	80	57	**59**
84				**61**	84	60	**61**
86				**61**	86	61	**61**
92				**65**	92	61	**65**
92				**66**	92	61	**66**
95				**66**	95	75	**66**
				71	63	82	**71**
				73	67	83	**73**
				78	67	86	**78**
				81	68	96	**81**
				86	76		**86**
				87			**87**
				88			**88**
				89			**89**
				91			**91**

Table D.1: The raw class marks for the respondants

Please note that in Table D.1 the last three columns are an amalgamation of the first five columns.

Appendix E

Anonymised impression survey results

Anon ID	How much time would you estimate you spent playing the game?	How many of the level sets would you estimate you completed?	I found the game to be easy overall.	I could complete at least the first few levels of each level set.	I had difficulty following and completing the tutorial level sets.	I accurately could predict what if a solution to a level was going to work before actually running it.	I had trouble making placements when there were a lot of symbols on the screen.	Watching the carrier as it moved helped me realise where I had made a mistake.	Being able to see the size and colour of a gem at all times helped me understand what was going on in my solutions.	Knowing the size and colour of a gem at a specific point helped me make decisions about how to approach a puzzle.	Seeing the size and colour of a gem change when a level ran helped me to better understand the function of specific symbols.	Playing the game helped me to more easily derive solutions for puzzles.	Playing the game gave me insight into the way programming languages work.	I'll find it easier to solve programming problems after playing the game.
1	<1h	40-60%	Agree	Agree	Disagree	Strongly Agree	Disagree	Neutral	Neutral	Neutral	Neutral	Agree	Agree	Agree
2	1-2h	<20%	Disagree	Strongly Agree	Agree	Neutral	Agree	Agree	Agree	Agree	Agree	Neutral	Strongly Disagree	Strongly Disagree
3	1-2h	40-60%	Agree	Agree	Neutral	Neutral	Neutral	Agree	Agree	Agree	Agree	Agree	Agree	Agree
4	<1h	<20%	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral
5	1-2h	<20%	Neutral	Agree	Strongly Agree	Neutral	Agree	Strongly Agree	Neutral	Neutral	Agree	Neutral	Disagree	Neutral
6	<1h	>80%	Strongly Agree	Strongly Agree	Strongly Disagree	Strongly Agree	Strongly Disagree	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree	Agree	Strongly Agree	Agree
7	4-6h	>80%	Agree	Strongly Agree	Strongly Disagree	Agree	Disagree	Agree	Agree	Agree	Agree	Agree	Agree	Agree
8	<1h	20-40%	Neutral	Strongly Agree	Agree	Neutral	Neutral	Agree	Agree	Agree	Agree	Agree	Agree	Neutral
9	1-2h	20-40%	Neutral	Agree	Agree	Disagree	Agree	Agree	Agree	Agree	Agree	Neutral	Neutral	Neutral
10	4-6h	>80%	Neutral	Strongly Agree	Strongly Dis-	Agree	Agree	Agree	Agree	Agree	Agree	Neutral	Neutral	Agree