DEVELOPMENT OF AN EDUCATIONAL GAME TO TEACH CONDITIONAL AND ITERATIVE CONTROL STRUCTURES

Amarnath Kakkar

A final year project submitted in partial fulfilment for the degree of Bachelor's in Computer Science and Mathematics with Honours
University of Bath

May, 2019

Development of an Educational Game to Teach Conditional and Iterative Control Structures

Submitted by: Amarnath Kakkar

COPYRIGHT

Declaration

Department of Computer Science
University of Bath

Supervisor: Dr. Alan Hayes

May, 2019

Abstract

To be written.

Table of Contents

List of Figures

List of Tables

Outline of Project

To be written.

Acknowldgements

To be written.

Chapter 1

Introduction

Chapter 2

Literature Review

2.1 Video Games

The video games industry has grown very rapidly in recent years, and is expected to continue to grow. The current market value is predicted as \$150 billion USD and is predicted to reach \$180 billion by 2022 (Newzoo, 2018). In 2006, video games were considered as one of the most popular forms of entertainment in the United States (Sherry et al., 2006; Ritterfeld and Weber, 2006). Currently, video games are considered a popular form of entertainment globally.

A video game can be defined as "a mental contest, played with a computer according to certain rules for amusement, recreation, or winning a stake" (Zyda, 2005).

2.1.1 Impacts

Initially, the majority of research on the effects of playing video games focused on the negative impacts, such as the potential aggression, addiction and depression from 'gaming'. However, researchers have argued that a more balanced perspective is needed (Granic, Lobel and Engels, 2014). Playing video games has been linked to an increase in perceptual, cognitive, behavioural, affective and motivational abilities (Connolly et al., 2012). Studies have also demonstrated that there are positive impacts from playing violent games, such as increased visuospatial skills (Ferguson, 2007).

2.1.2 Uses

Video games are now used for a wide variety of reasons. They are becoming increasingly useful in the global education and training market. Apart from entertainment, video games are being used in: military, government, education, corporate and healthcare (Susi, Johannesson and Backlund, 2007).

2.2 Educational Games

The term 'Serious Game' can be used to describe an educational game. A serious game refers to a game that has an educational purpose and is not intended to be played primarily for entertainment (Abt, 1970).

Serious games became an established academic field of study in 2007, after establishment of The Serious Games Institute (Wilkinson, 2016). The market value of the serious games industry in 2016 was predicted at $1.5 billion USD, and is predicted to reach $9 billion in 2023 (AlliedMarketResearch, 2017).

2.2.1 Definition

There is currently no singleton definition for term 'Serious Game'. Susi, Johannesson and Backlund argue that, groups and individuals define the term depending on their perspectives and interests, and that there are a wide variety of groups and individuals focusing on different issues (Susi, Johannesson and Backlund, 2007). The first recorded definition of the term was set out in 1970 by Abt (Wilkinson, 2016), who defined it as follows: "Games that have an explicit and carefully thought-out educational purpose, and are not intended to be played primarily for amusement. This does not mean that serious games are not, or should not be, entertaining." (Abt, 1970). In 2005, Michael and Chen built on this definition to come up with the definition that they are "Games that do not have entertainment, enjoyment or fun as their primary objective" (Michael and Chen, 2005). This definition suggested that serious games are not limited to only educational purposes. The commonly agreed upon definition closely matches the definition by Michael and Chen (see Susi, Johannesson and Backlund, 2007).

Another definition that is often referred to was set out by Zyda (see Susi, Johannesson and Backlund, 2007). This definition however contradicts the one set out by Michael and Chen (Susi, Johannesson and Backlund, 2007). In contrast, Michael and Chen argue that serious games should not, have entertainment or fun as their primary objective. On the other hand, Zyda expressed that the entertainment component of the game should come first, and also that the story of the game is more important than the pedagogy (Zyda, 2005). His definition is as follows: "a mental contest, played with a computer in accordance with specific rules, that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives" (Zyda, 2005). However this definition also suggests that serious games can only be digital (Djaouti et al., 2011).

Since there are a variety of definitions, for this dissertation, I will use the one provided by Abt and work entertainment around the primary purpose of the game - to teach.

2.2.2 History

Educational games have arguably existed since the 7th century. Among the oldest is the board game 'Chaturgana', which is argued by historians to be the precursor to chess (Wilkinson, 2016). The aim of the game was to teach officers to become better planners for battles (Wilkinson, 2016). Another board game created more recently - in the 20th century was 'Landlord's Game'; a precursor to monopoly. It was designed to illustrate the dangers of capitalist approaches to land taxes and property renting (Wilkinson, 2016). So we can see that, games designed to educate, have existed for a long time.

The interest in digital educational games, has been observed since 1967, when the first educational program was developed, 'Logo Programming' (Hayes and Games, 2008) . Logo programming was an environment that allowed players to utilise the programming language LOGO in order to learn mathematics (Feurzeig et al., 1969). Logo was popular among schools in the US (Lehrer, 1986), and it became a key part of educational strategies research (Hayes and Games, 2008).

Academic interest in games that could be used for purposes apart from entertainment emerged emerged in 1970 by Abt, in his book *Serious Games* (Breuer and Bente, 2010). The rise in digital games around this time created an opportunity for developing serious games (Wilkinson, 2016).

However, serious games did not gain much traction until 2002, when a game developed by the US Army, became hugely popular. 'America's Army' was developed as a training and recruitment game for the military. It is now considered as the forefront of modern serious games (Zyda, 2005; Wilkinson, 2016). In conclusion the potential of using games as educational tools has been demonstrated for a long time, and is now being further researched and developed, focusing on training and education in a number of different industries (Wilkinson, 2016).

2.2.3 Benefits

Serious games have become an interesting area for multidisciplinary academic research (Breuer and Bente, 2010). There are interests from fields such as psychology, computer science, pedagogy, sociology and cultural studies (Breuer and Bente, 2010). Many studies have looked into and discussed the benefits of serious games in educational contexts, and I will discuss some of these below.

2.2.3.1 E-learning

E-learning can be defined as an approach to teaching and learning, based on the use of electronic media and devices (Sangrà, Vlachopoulos and Cabrera, 2012). Thus, digital educational games can be seen as a type of e-learning.

Educational games have inherent beneficial properties. For instance, they are able to provide information on demand and just in time, and in the context of actual use and people's purposes and goals, something that does not often happen in schools (Gee, 2003).

Other properties of e-learning include: ease of accessibility; can be used in absence of teachers or instructors; provide opportunities for relations between learners, helping eliminate the potential of hindering participation; low cost per person served; allows self-pacing - allowing student to study at their own pace; high level of interactivity; ability to use attractive graphics, and is an engaging and entertaining activity (Arkorful and Abaidoo, 2015; Girard, Ecalle and Magnan, 2013).

In a study carried out on what university students thought about e-learning, students reported that they expected e-learning to be an integral part of the learning process within higher education (Connolly et al., 2012). Thus, the use of a digital game in higher education, may not be an unfounded concept to students.

2.2.3.2 Learning through Games

Games can be a great learning environment (see Prensky, 2003; Gee, 2003). Despite the vast research on the negative impacts of gaming, playing video games identified for imparting beneficial skills such as, visual attention, spatial skills, problem solving skills and creativity (Granic, Lobel and Engels, 2014).

When players first begin a new game, they first learn the rules and the controls of the game, then use this newly acquired knowledge to complete objectives or levels. As players progress through the game, it becomes more difficult and calls for them to use better skills and knowledge for them to move to the next stages of the game. Vygotsky coined the term *the zone of proximal development*, where students gain a higher degree of learning when they are presented with tasks which are just beyond their current level of ability but may require some help to complete (Vygotsky, 1978).

Today's learners have grown up immersed in digital technology (Prensky, 2001); Prensky calls this generation of people 'Digitial Natives', because they have spent long periods of time playing video games (Prensky, 2003). In 2016, 2.5 billion people were reported to actively play video games worldwide (AAStocks, 2016), and a year later, roughly 57% were aged between 10 and 35 (Newzoo, 2017). Therefore, this generation will be more receptive to computer-based learning (Girard, Ecalle and Magnan, 2013).

Playing games is naturally a fun and pleasurable activity (Prensky, 2001), and research has shown that fun is important for the learning process, as learners can be more motivated and willing to learn (Bisson and Luckner, 1996; Cordova and Lepper, 1996). Learners' enjoyment of a game has also been shown to improve learners' acquisition of knowledge (Giannakos, 2013).

2.2.3.3 Effectiveness of Learning through Games

Evidence of learning through educational games has been demonstrated (Connolly et al., 2012; Wouters et al., 2013; Girard, Ecalle and Magnan, 2013). However, the effectiveness of educational games is undetermined (Connolly et al., 2012; Girard, Ecalle and Magnan, 2013). There are also different viewpoints when it comes to determining their effectiveness. Two such meta-analyses, Wouters et al. (2013) and Girard, Ecalle and Magnan (2013), explored at the effectiveness of learning through games, and made different conclusions. These studies are compared in Table 2.2.1.

Table 2.2.1: Comparison of meta-analyses on the effectiveness of learning through games

	Wouters et al. (2013)	**Girard, Ecalle and Magnan (2013)**
Types of Games	Serious games	Serious games & video games
Measuring	Learning, retention and motivation	Learning and engagement
Learning Outcomes	Knowledge or skill acquisition	Knowledge or skill acquisition
Instructional Domain	Biology, maths, language or engineering	Various (including: academic knowledge, cognitive skills, professional knowledge, cancer therapies)
Studies Published Between	1990 - 2012	2007 - 2011
Experimental Design of Studies	Posttest or pretest-posttest	Atleast pretest-posttest
Studies Reviewed	39	9
Age Range of Studied Population	Wide range	9 - 47
Results support effective learning through games	Yes	Inconclusive

Wouters et al. found that serious games were more effective in terms of learning and retention when compared to conventional teaching methods, but not more motivating. On the other hand, Girard, Ecalle and Magnan found that only a few of the games resulted in improved learning, with the others having no difference when compared to traditional methods of teaching. A point to note is that in Wouters et al. (2013), serious games were compared to lectures, reading, drill and practice, or hypertext learning environments, whilst in Girard, Ecalle and Magnan (2013), they were compared to face-to-face lessons, pencil-and-paper studying or no studying at all. The former is a more modern way of teaching, whereas the latter is very limited.

Wouters et al. evaluated a broad spectrum of studies, whereas Girard, Ecalle and Magnan only evaluated randomised control trial studies. Wouters et al. argued that if they only considered the studies with randomised samples with a pretest-posttest design, similar to Girard, Ecalle and Magnan study, the positive effects in favour of serious games may disappear.

In conclusion, there is a need for more empirical research to determine the effectiveness of serious games, and this is being addressed (Connolly et al., 2012).

2.2.4 Examples

There are many existing educational games designed to teach various topics. I will cover some educational games and interesting technologies that help further programming skills.

Scratch

Scratch is a block-based visual programming language and an online community for sharing, discussing and 'remixing' one another's projects. Projects can range from creating your own interactive stories, games, animations or simulations, which users can share with one another online (Resnick et al., 2009). Today, Scratch has accumulated almost 40 million users, with the core audience between ages 8 and 16, which have created and shared more than 40 million projects (Scratch, 2019).

The Scratch language is based on a collection of programming blocks that can be snapped together to create programs. Scratch blocks are shaped in a way such that they can only fit with other blocks to make syntactic sense (see Figure 2.2.1). The creator of Scratch and fellow researchers argued that the social aspect of Scratch was important for it to succeed, with a user claiming that she learnt a lot about different kinds of programming by looking at, downloading, and modifying the scripts from other peoples games. Users can also work together on projects, helping them develop their collaborative and team leadership skills (Resnick et al., 2009).

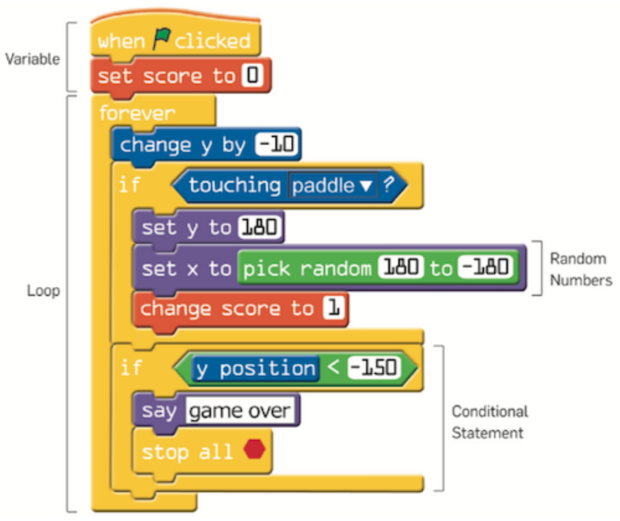

```
when 🏴 clicked
set score to 0
forever
  change y by -10
  if <touching paddle ▾ ?>
    set y to 180
    set x to (pick random 180 to -180)
    change score to 1
  if <(y position) < -150>
    say game over
    stop all 🛑
```

Figure 2.2.1: An example Scratch script (Resnick et al., 2009)

Scratch has also been used in higher educational contexts (Resnick et al., 2009). In a study conducted by Malan and Leitner (2007), Scratch was used at Harvard's summer school to introduce students to programming. After a total of five hours on Scratch, students transitioned onto Java for the remainder of the course. Malan and Leitner found that more than 75% of students felt that using Scratch, was a positive influence with their transition onto Java; a more complex syntactical language (Malan and Leitner, 2007). It is clear from the student responses that Scratch helped develop their computational thinking skills. This is often hard to do in languages like Java as students can feel overwhelmed trying to learn the complex syntax, which can hinder their ability to learn to program (Koulouri, Lauria and Macredie, 2014).

Prog&Play

Prog&Play was developed to tackle the issue of student motivation in computer science courses in higher education. Prog&Play is a 3D real-time strategy game designed to strengthen programming skills. It was built on top of an existing open-source multiplayer game, because of the potential advantage of the game already being robust. The developers of Prog&Play created an API that enabled students to interact with the game through programmable commands, as the original game did not contain any programming (Muratet et al., 2011).

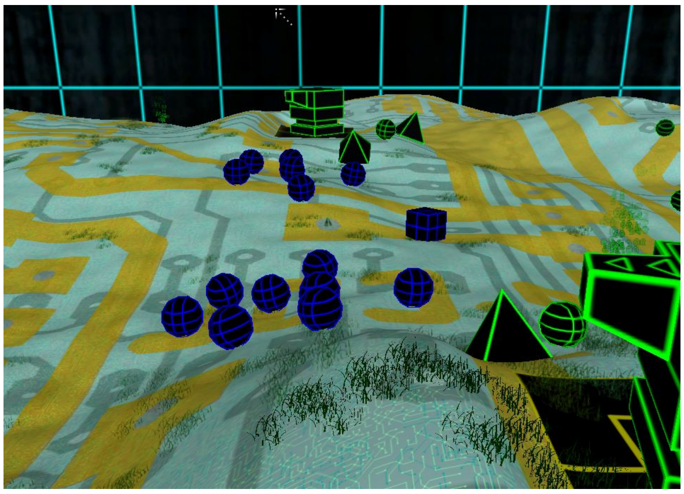


Figure 2.2.2: A screenshot of the game Prog&Play (Muratet, 2012)

Prog&Play introduced variables, functions and conditional and iterative control structures. The game is based on a war between the factions, 'Systems', 'Hackers' and 'Networks' (see Figure 2.2.2). Players choose a side and give orders to their units through a set of commands. The developers also created a single player mode, where students could be gradually introduced to learning topics and learn how to play. The multiplayer aspect then allowed students to create their own programs and compete with each other (Muratet et al., 2011).

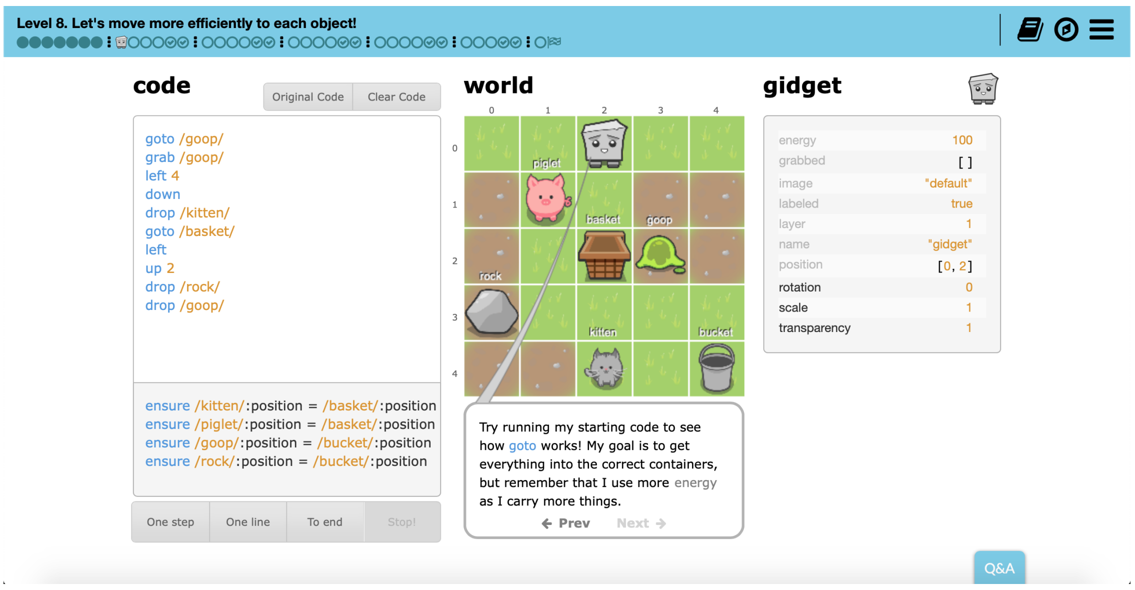
Through examinations conducted by the teachers, Muratet et al. found that students who played the game achieved better marks than those who did not. Many students who played the game went on to choose a computer science unit the next semester, compared to those that did not play the game at all. Interestingly, a majority of students preferred the programming version, Prog&Play, to the original game which did not require any programming (Muratet et al., 2011).

Gidget

Gidget is a web-based game developed as part of a study to improve learning in novice programmers. The study tested the effects of personifying programming tool feedback. One such personification was to programming errors that were displayed to users (Lee and Ko, 2011).

In the case of Gidget, you have have to work with a damaged robot, also named Gidget, to help clean up a city and protect its animals after a chemical spill. The robot, Gidget, acts as a companion and feedback tool. It uses personified language, takes the blame for syntax and runtime errors, and has an emotional face (Lee and Ko, 2011). Lee and Ko claimed that this would change the role of the conventional feedback tool often perceived as an authoritative figure, which can be off-putting, to a collaborator needing assistance. The game teaches the design and analysis of basic algorithms in a language designed specifically for the game. Players have to debug the code produced by Gidget to complete the levels (Lee and Ko, 2011).

Lee and Ko carried out a study with 116 participants, using two versions of the game. One version was with a personified Gidget robot (see Figure 2.2.3), and the other with a Gidget that had a faceless screen that provided impersonal feedback. The results revealed that the group using the personified Gidget, completed more levels in roughly the same amount of time in comparison to the group using the faceless Gidget, suggesting that the personification of feedback had a positive effect on participations' motivation to play. Although, the personification of Gidget did not affect the enjoyment felt playing the game (Lee and Ko, 2011).



```
goto /goop/
grab /goop/
left 4
down
drop /kitten/
goto /basket/
left
up 2
drop /rock/
drop /goop/
```

```
ensure /kitten/:position = /basket/:position
ensure /piglet/:position = /basket/:position
ensure /goop/:position = /bucket/:position
ensure /rock/:position = /bucket/:position
```

Try running my starting code to see how goto works! My goal is to get everything into the correct containers, but remember that I use more energy as I carry more things.

energy	100
grabbed	[ ]
image	"default"
labeled	true
layer	1
name	"gidget"
position	[0, 2]
rotation	0
scale	1
transparency	1

Figure 2.2.3: A screenshot of the game Gidget (Gidget, 2019)

2.3 Teaching Conditional and Iterative Control Structures

In programming language design and implementation, ’control structure’ refers to those constructs which determine the sequence of execution, of operations and statements within programs (Riccardi, 1981). This definition is still relevant today (see Wikiversity, 2019). There are three types of control structures; sequential, conditional, and iterative (Leppänen, 2007). A sequential structure is execution of a code line by line. A conditional construct is used to make decisions, for example, *if* and *if else* statements in JavaScript. An iterative construct is used for looping, for example, *for* and *while* loops in JavaScript.

These control structures are likely to be taught in larger introductory units, which cover a wide range of fundamental concepts (ACM-IEEE, 2013). For example, at the University of Bath, conditional and iterative structures are taught in semester 1 of year 1 of a computer science undergraduate degree (UniversityOfBath, 2018).

2.3.1 Methods

At university, introductory programming units are conventionally structured courses based on lectures and practical laboratory work. Students learn about programming concepts through the programming language being taught (Robins, Rountree and Rountree, 2003; ACM-IEEE, 2013). Robins, Rountree and Rountree suggest that this approach is popular due to the importance of the programming knowledge gained from the programming language, and the sheer volume and detail of language related features that can be covered.

Another method for learning iteration and conditional constructs, which is becoming increasingly popular is online resources. These resources include tutorial websites, such as Codeacademy and Khan Academy, which have accumulated millions of users, block-based programming environments such as Scratch and Alice, which provide creative visual environments, and educational games (Lee and Ko, 2015).

Both traditional forms of learning and e-learning, guide the student through the learning experience and often set the students practical work to complete. However there are a few differences between these methods. In contrast, traditional methods of teaching centre on instructors who have control over class content and the learning process, whereas online resources, provide a learner-centred, self-paced learning environment (Zhang et al., 2004). Zhang et al. (2004) goes are as far to say that e-learning could replace traditional classroom learning. Nevertheless, using e-learning to complement the learning process can make learning more effective and improve the learning experience (Zhang et al., 2004; Concannon, Flynn and Campbell, 2005)

2.3.2 Issues

It is generally accepted that learning to program is a difficult task, and there are several problems associated with this (Koulouri, Lauria and Macredie, 2014). For leaners, the overhead of learning the syntax and semantics of a language at the same time, difficulties in combining new and previous knowledge, and developing their problem-solving skills, all add to the complexity of learning how to program (Koulouri, Lauria and Macredie, 2014).

There is also concern over the high drop out and failure rates of first and second year University students in computer science courses. This is potentially associated with considerable failure rates in introductory programming units (Koulouri, Lauria and Macredie, 2014). However, researchers have argued that there is little to no high-quality empirical data to support these claims (Bennedsen and Caspersen, 2007; Watson and Li, 2014). In two different studies; Bennedsen and Caspersen (2007), and Watson and Li (2014), both found that the failure rate of introductory units were 33% and 33.3%, from an average of 63 and 51 institutions respectively. Both studies argued that these rates were not "alarmingly high", but there is considerable potential for improvement (Watson and Li, 2014).

Koulouri, Lauria and Macredie (2014) carried out 4-year study and found that certain implementations of introductory units, could improve student learning performance. One finding was that programming proficiency of novice programmers is dependent on the teaching approach of an introductory programming unit. Introductory programming units are also considered difficult (Watson and Li, 2014). In conclusion, choosing an approach that is engaging and motivating to teach fundamental programming concepts, is important.

2.4 Educational Game Design

It is not sufficient to just assume that all forms of games are equally suitable for learning, and that simply presenting material in a game-like setting, will increase the quantity and quality of learning (Breuer and Bente, 2010). Designing educational games requires a focus that is different from general game design; otherwise, we may end up designing fun games with little or no learning value (Barnes et al., 2007).

Breuer and Bente (2010) argue the ideal educational game combines entertainment and learning in a way that the players/learners do not experience the learning part as something external to the game. This idea of stealth learning should inform any approach to designing, using and evaluating educational games (Breuer and Bente, 2010). However, combining compelling and interactive design elements with specific curricular content that aims to retain learner's interest and attention, is a difficult task (Prensky, 2003).

2.4.1 Games and Learning

Researchers have looked into the many factors that influence learning (Roungas and Dalpiaz, 2015). Below are some of these factors that influence learning in a positive way, and how these are implemented in games. Roungas and Dalpiaz (2015) argue that these following factors are important and should be considered when designing serious games.

2.4.1.1 Motivation

Motivation is a key factor that drives learning. Without motivation, learning stops (Gee, 2003). A key factor of motivation is interest (Rousseau, 1762) (as cited in Roungas and Dalpiaz (2015)). Learning is effective when learners are interested in the material being taught and how it is presented.

Games are natural sources of motivation. Games that bring about an optimal balance between challenge and frustration for players, is a very motivating state to be in (Gee, 2003). In his theory of flow, Csikszentmihaly describes that some leaners can become so focused in and absorbed by an activity that they lose track of time and completely ignore other tasks (Csikszentmihalyi, Abuhamdeh and Nakamura, 2014).

Researchers have looked into the intrinsic motivational properties of video games, that is, player motivation that arise due to factors within the players themselves. For example, pleasure and desire to develop a skill are forms of intrinsic motivations (Roungas and Dalpiaz, 2015). Malone and Lepper (1987) investigated these properties of games, and presented a set of aspects of games that would make game-based learning more fun and engaging. The work by Malone and Lepper is often referred to, but Whitton (2011) noted that Malone's work is not as valid today or as valid when considering adult engagement. With this, Whitton described five aspects that lead to engagement specifically in games targeted for higher education: challenge, control, immersion, interest and purpose.

2.4.1.2 Repetition

Repetition is considered crucial for learning, but some modern researchers do not consider it very important. However, it is deemed useful for learning skills such as programming (Roungas and Dalpiaz, 2015), where it could be used for learning programming syntax and function names.

Repetition is very common in video games. It allows players to learn, memorise and potentially master some rules or controls of the game. Once players are familiar with these skills, the game then introduces a new problem, which requires players to integrate their old skills with new ones. These new problems are practised until a problem requiring a different solution comes along. This cycle is repeated throughout the game (Gee, 2003).

However, repetition needs to be managed, as it could lead to boredom. To avoid this, Roungas and Dalpiaz (2015) suggest modifying the content of the task after each repetition. This can be achieved in games by slowly introducing new rules or controls that build upon previous game knowledge, or, by framing the task in different scenarios. Roungas and Dalpiaz (2015) also suggested managing how frequent the task is presented. A task repeated too often could lead to boredom, or a task repeated at very long time intervals, could reduce the learning benefits from repetition.

2.4.1.3 Rewards

The use of rewards and punishments in learning has various impacts. Firstly, rewards allow a learner to link learning with something pleasant. Secondly, rewards are form of extrinsic motivation, that is, learners are motivated by factors external to themselves. For example, achieving high grades, winning trophies or recognition by peers are all forms of extrinsic motivation (Roungas and Dalpiaz, 2015). Thus attempting to seek such rewards may give learners motivation to continue their engagement with the activity. Finally, rewards were linked to increased learning performance in a study that used rewards in an educational game (Filsecker and Hickey, 2014).

Wang and Sun (2011) goes as far as to say, rewards can foster intrinsic motivations, specifically, fun that arises from achieving rewards. An increase in intrinsic motivation in learners, has shown that learners are more deeply involved in activities and are more confident (Cordova and Lepper, 1996).

There are a variety of ways rewards can be expressed in video games, the following are some examples: score system, experience point system, item granting, resources, achievements, instant feedback, plot animations and unlocking mechanisms (Wang and Sun, 2011). However, the relevancy and frequency of rewards needs to be managed, otherwise they can lead to diminishing returns; rewards should be suitable to the target audience, and like repetition, the frequency of rewarding is important. Rewarding too often can cause players to lose interest, or rewarding very little can lead players to lose motivation to play (Roungas and Dalpiaz, 2015).

2.4.2 Game Elements

Games have several characteristics. Roungas and Dalpiaz (2015) argue the following characteristics are important for serious game design. These characteristics are aimed at keeping games engaging, exciting and educational.

1. **Rules**. Rules are essential to a game. They describe how the game works and limit the players' actions (Roungas and Dalpiaz, 2015). Whilst surveying players of introductory programming games, Barnes et al. (2007) gathered that rules should be provided and accessible throughout the game. This would allow players to refer back to the rules during the game

in case they needed reminding or help.

2. **Goals**. Goals allow players to judge their performance. They also act as a motivation for players (Roungas and Dalpiaz, 2015). (Barnes et al., 2007) found that goals should also be provided and accessible throughout the game. They must be clearly tied to in-game feedback, such as avatar health or score, that either motivates or penalises players (Barnes et al., 2007).

3. **Challenge**. Challenge leads to engagement, if, tasks are optimally balanced between challenge and skill (Whitton, 2011; Csikszentmihalyi, Abuhamdeh and Nakamura, 2014). Challenges need to be carefully designed in order to not be too boring or too difficult (Roungas and Dalpiaz, 2015).

4. **Feedback**. Feedback allows players to assess their performance (Roungas and Dalpiaz, 2015). It leads players towards correct answers if their assumptions are wrong, and thus creates learning opportunities. However in programming, feedback in the form of errors can be quite discouraging to novice programmers (Lee and Ko, 2011). Thus care should be taken when designing feedback in a beginner programming game. Feedback should be presented clearly and on time (Roungas and Dalpiaz, 2015).

Chapter 3

Requirements Specification

The literature review highlights several key areas of study that were used to inform the requirements specification. The following parts of the literature review will be considered; benefits of educational games; issues of teaching iterative and conditional control structures; the elements from the example of educational games that had a positive impact, and finally educational game design.

The requirements are split into functional and non-functional sections, and each requirement is given an importance rating of: high, medium or low, which refer to being essential, conditional or optional for the system respectively.

The aim of the current study is to develop an educational game to teach iterative and conditional control structures. This will be achieved through the following requirements:

3.1 Non-Functional Requirements

3.1.1 **Game must improve novice programmers' knowledge on iterative and conditional control structures.** This is the main aim of the study. A key objective of an educational game is to educate the user.
Priority: High

3.1.2 **Game must not assume player has any existing programming knowledge.** The game should be a useful learning tool for anyone wanting to learn about iterative and conditional control structures, regardless of prior knowledge.
Priority: High

3.1.3 **Game must be easily accessible from home, work or school.** A key benefit of a digital educational game, is the ability to access it and learn from it anytime or anywhere.
Priority: High

3.1.4 **Game should not require complex installation.** The game should be easily and readily accessible, and be quick to start playing.
Priority: Medium

3.1.5 **Game should be fun to play.** Fun is a key aspect of an educational game as it can helps motivate players (Cordova and Lepper, 1996) and improve their learning performance (Giannakos, 2013).
Priority: Medium

3.1.6 **Player should be engaged throughout the game.** Engagement is key in learning.
Priority: Medium

3.1.7 **Player should feel challenged.** Challenge is a key aspect that leads to player engagement (Whitton, 2011).
Priority: Medium

3.1.8 **Game should have a reasonable difficulty curve.** Players should be able to get a reasonable understanding of a concept before being introduced to new ones. The game should also have a balance between challenge and skill.
Priority: Medium

3.1.9 **Game may foster player motivation to learn more complex programming.** Student motivation in introductory programming courses is very low and can be improved (Koulouri, Lauria and Macredie, 2014).
Priority: Low

3.2 Functional Requirements

3.2.1 **Game must at least introduce *if* statements and *for* loops.** These constructs can be considered essential iterative and conditional control structures, and so the game must cover these.
Priority: High

3.2.2 **Game must have levels with their own clear defined mini goals.** Clear defined goals will tell the player what to do each level. This can help to introduce iterative and conditional constructs.
Priority: High

3.2.3 **Game must give players some control over choice of action.** Control over choice of action is a key aspect that leads to engagement (Whitton, 2011).
Priority: High

3.2.4 **Game must give feedback to player solutions.** Feedback is important as it allows players to assess their performance and deal with any mistakes. Thus creating learning opportunities.

Priority: High

3.2.5 **Game must use some form of in-game rewards and punishments, such as score and hit points.** In-game rewards and punishment foster player engagement (Roungas and Dalpiaz, 2015).

Priority: High

3.2.6 **Game must have a help section.** Whenever players are stuck, they should be obtain help from the help section.

Priority: High

3.2.7 **Game rules must be accessible throughout the game.** Players must be able to refer back to the rules throughout the game.

Priority: High

3.2.8 **Game should give feedback in a personal and constructive way.** Personal and constructive feedback may improve player motivation (Lee and Ko, 2011).

Priority: Medium

3.2.9 **Game should have a tutorial to introduce the game.** A tutorial will be able introduce the game in an engaging and interactive way.

Priority: Medium

3.2.10 **Game should be web-based, and playable in three most popular web browsers.** These are Chrome, Safari and Firefox (StatCounter, 2019). Allows the game to be easily and widely accessible.

Priority: Medium

3.2.11 **Game may provide tips to the player about individual level.** In case player is ever stuck on a level and the help section does resolve the player issues, tips should help the player.

Priority: Low

3.2.12 **Game may save players progress.** Saving game state would allow players to exit the game for any reason, and return any time they wish.

Priority: Low

Chapter 4

Design

Since the start of the project, the idea for the genre of the game was a 2D maze adventure. Researchers have argued which genres of games are best suited for educational purposes (Moreno-Ger et al., 2008), however most of the discussion on educational game design does not revolve around the game genre, but instead on which game elements, pedagogical concepts and player feelings aroused by the game are most important. These aspects were discussed in the literature review and were used to inform and influence the requirements specification. This chapter aims to indicate how the problem, as specified within the requirements, is analysed to create a potential solution.

4.1 Core Idea

The core idea of the game is a character stranded in a dungeon, and the user must write pieces of code to control the character and help him escape. To do this, the user is given a set of pre-defined functions to use to control their character. These pre-defined functions will simply be referred to as 'functions' throughout this study. Throughout the game, users are slowly introduced to *if* statements and *for* loops, which they then must use with the functions to create pieces of code. The aim is for users to initially create simple pieces of code, then move onto creating more complex structured and efficient code as they progress.

To help discuss the design, the goals of the game are split into two. The first goal, and also the aim of this study, is to educate users on certain iterative and conditional control structures. These will be referred to as the educational goals. The second goal is to help the in-game character escape. This will be referred to as the game goal.

4.2 Web-based

A web-based design was adopted to develop the game. A web game has several advantages over other forms of video games. Namely, these are:

- Accessible on most web browsers without having to create different versions of the game for each browser. This is cost-effective and reduces hassle for users.
- Accessible through a number of different web-based devices, without having to create different versions of the game or install the game onto each different device. This allows the game to be accessible in more places.
- Ease of accessibility, given that computers with an internet connection are ubiquitous. Only requires a browser to play.
- Requires no additional download, installation or update to plug-ins. Reduces the complexity needed for users to access the game.
- Easy to update web applications once they are deployed. Requires no update from user to get the latest version of the game, apart from perhaps updating their web browser to the latest version.
- Can achieve greater levels of interoperability compared to desktop environments (Roungas and Dalpiaz, 2015).

4.3 Game Mechanics

Game mechanics can be defined as the methods invoked by agents for interacting with the game world (Sicart, 2008). In the context of this game, the agents are the players and the methods are those elements of the game, that the user interacts with either directly or through their actions, that effect the game state. The following subsections will discuss these elements in more detail and the justification for their adoption.

4.3.1 Functions

Commonly when using *if* and *for* constructs, statements are defined within the construct to be executed. For this game, functions that help the in-game character, were decided to be used as such statements. Furthermore, the use of functions to control in-game characters had a positive impact on player engagement in the educational game Prog&Play.

Since the idea was for the character to be stranded in a dungeon. One essential function would be to get the character to move. However, just using one function throughout the game may get a bit boring. Thus in attempt to add to the dungeon feel, another function was designed to shoot. The dungeon would be filled with enemies, which the character would have to shoot to get through to the exit. Walk and shoot would be all the functions the player could use to directly interact with their character.

Two more functions were designed to be used as conditions within conditional constructs. Specifically these are, functions that would return true or false based upon facts about the game state: if a wall is nearby, or, if an enemy is nearby. Using these in conjunction with the walk and shoot commands, would give users flexibility to come up with their own unique solutions to a level. Thus essentially giving users some control. Learning to program occurs efficiently, when the learner is able to use trial and error, come up with their own solutions and learn from their mistakes.

Throughout the game players would have to use these functions with *if* statements and *for* loops to help achieve the game goal, in turn helping them achieve the educational goals. Thus these functions can be seen as the bridge between the entertainment aspect of the game and the educational aims.

However as discussed in the literature review, care needs to be taken when trying to introduce programming to novice programmers. They can often struggle with the complexity of the syntax and semantics of the programming language. To tackle this issue, the aim is to dissipate the introduction of these functions and constructs throughout the game, which will be discussed in more detail in the level design section. This is also why only four functions were designed, which all take one of four parameters: left, right, up or down. This would give players the freedom to create their own solutions, whilst keeping the complexity required to play the game low.

4.3.2 Scoring

Score is an essential aspect in many video games. It provides players feedback based on their performance, and create opportunities for them to want to improve. It also brings about a sense of challenge, by wanting to achieve a good score. In turn fuelling player motivation and engagement.

Initially the idea was to implement a score system that would score based on players' code. This however would be a complex system to develop, as there is not one fixed way to complete a level and hence not all code solutions will look the same. Instead, a scoring system that scored players based on the time taken to complete a level is more suitable.

The design of this scoring system is as follows:

1. Players get a fixed score at the start of the game, which would immediately begin to decrease.
2. Upon completion of a level, players' score will increase by a fixed amount, which will then continue decreasing.
3. This would repeat until the player completes the final level, upon which they would be

presented with their final score.

This style of scoring system would potentially challenge players to come up with solutions quickly in the aim for a high score. However a drawback may be that players get frustrated if they feel forced to come up with a solution quickly. To tackle this, the fixed amount a score would increase by would be a large number, and the score would decrease very gradually. Thus players are more likely to end up with a high score, which potentially may decrease any pressure to rush. In conclusion, the overall aim for this style of scoring system is to increase player engagement and bring a sense of challenge.

4.3.3 Hit points

Since the design of the game was for the character to be able to shoot, specifically at enemies. It is viable that the character can get 'hurt' by such enemies. This would involve the characters hit points decreasing. Hit points are generally present in games that involve shooting, so again it is a viable mechanic for this style of game.

The character will be given a fixed hit point at the start of the game, which will only decrease if the character walks into any enemies. This would only occur when players have not created an optimal solution for the current level. Upon losing all their hit points, or in other words if their character 'dies', the character would be reset to the initial position at the start of the same level. The player will also lose some points from their score. The users code will not reset, as to give them a chance to identify the issues with their code, amend it and try again.

This style of mechanic would give players the incentive to want to avoid such scenarios by planning and creating well-designed and thought-out code. The more the player thinks about their code, the more understanding they will develop about their code and the constructs they are working with. Specifically how these constructs work. The idea to reset the character upon death adds to this by giving players a chance to figure out where they went wrong, create a different solution and further their understanding.

4.4 Level Design

Level design is an important aspect of the design stage. The main decisions to be made for each level are which iterative and conditional constructs to introduce. The main considerations are: the difficulty of each level, and the pace of teaching.

The levels should not be too difficult that users can not complete, and nor too easy that the users get bored. As discussed, the levels should bring about an optimal balance between challenge and frustration. So the following concepts were considered to introduce: *if*, *if else* and *else* statements, *for* loops, nested *for* loops and a combination of all these.

Thus the user is not only is introduced to the basic constructs, but learns how these can be combined to create larger programs. Combining these constructs allows to produce game levels with varying levels of complexity. Initially the user is presented with basic *if* and *for* constructs, which are slowly developed on throughout the game, until they have to use most, if not all, of these constructs in order to complete a level. However care needs to be taken, that the users are familiar with certain concepts before being introduced to newer more complex ones.

Pace of teaching should be designed as to not overwhelm users with too much information, and not bore users by going very slowly. As soon as players are familiar with certain concepts, they should be introduced to new or more complex ones. To help make certain that users were as familiar as possible with these concepts, three types of levels were introduced: to complete unfinished code, debug given code, or create your own solution. These also aim to provide more functionality to the game and perhaps make the game more entertaining, as just doing only one of three objectives throughout the game may get boring.

These three types of level allow the same constructs to be reused in different scenarios. Repetition is key for learning, and these levels help provide this. At the start of introducing a new construct, users will first have to tackle a level that requires to complete unfinished code, as this can be made the most easiest of the three. This type of level, will allow the user to familiarise themselves with the construct. The next type of level they will face is to debug code. This goes a bit beyond completing unfinished code, as users can familiarise themselves with the syntax of the construct. Finally, the user can then be presented with a level that requires their own solution. This type of level should hope to bring together all the previous knowledge they gained and apply it to create a unique solution. The proposed structure of the levels can be seen in Table 4.4.1.

Table 4.4.1: Design of level structure of the game

Level	Type of Level	Programming Concepts Required
Tutorial	N/A	None
01	Unfinished code	for loop
02	Debug	for loop
03	Unfinished code	if statement
04	Unfinished code	for loop, if and else statements
05	Create your own	for loop and if statement
06	Debug	for loop, if and else if statements
07	Unfinished code	nested for loop
08	Create your own	Most

4.4.1 Tutorial

A tutorials role is to introduce the game to the player. Tutorials commonly entail the introduction of the game goals, game mechanics and rules. They can sometimes also have a walkthrough which leads the user through an example level. However, if a tutorial is too long or contains too much information, players might quickly get bored and lose any motivation they had, to play. Thus which information to present in the tutorial was carefully considered.

The tutorial will comprise of two sections. The first will be a dialogue that explains the educational goals, as they are the main objective of the game, the game goals, game mechanics excluding the functions, and rules. It would be more useful to introduce functions to players during the game when they are actually required. Constructs will also be introduced during the levels for the same reason. The second section will be a walkthrough of an example level. The example level will have already have code required to complete the level, the user will be instructed to execute it. This gives players a feel for how the functions and character interact.

Since the game can be played more than once by the same user, the tutorial will give users the option to skip and continue onto the main game. This is so users who already know how to play, do not have to spend unnecessary time recapping material already known. However throughout the game, the game interface will have a help section that contains a brief summary of the information presented in the tutorial.

4.5 User Interface

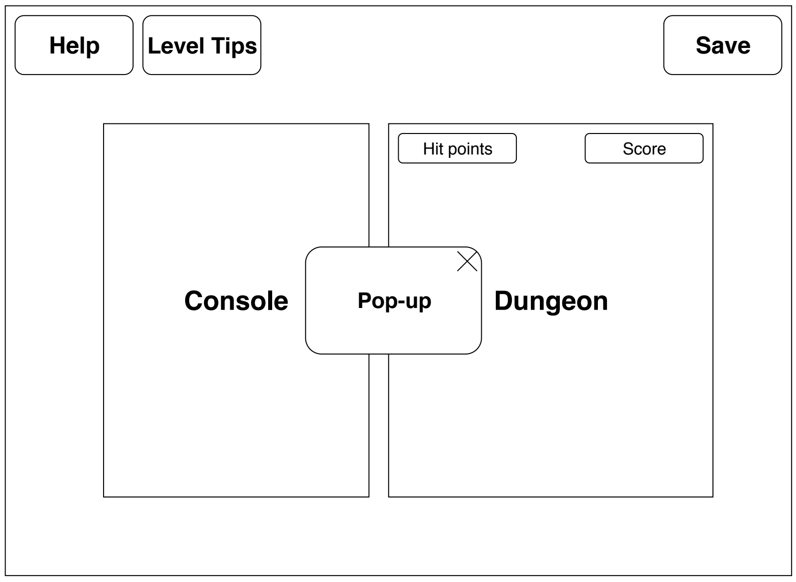


Figure 4.5.1: Design of game layout for Programmer's Dungeon

An important aspect of the interface is for it to be aesthetically pleasing. It is the first thing a user sees, so it must be designed to encourage motivation. Figure 4.5.1 shows the design layout of the game. The elements in the design were selected in order to satisfy the requirements. The description and purpose of each element is explained below:

- **Console.** An updatable console for players write and modify code. Console will also have an execute button to run code. Level requirements will also be displayed here.
- **Dungeon.** Displays the layout of the current level which includes: character, any enemies and the exit. Character will carry out the code executed in the console.
- **Pop-up.** This will only ever pop-up to introduce and explain unintroduced constructs and functions as they are required.
- **Hit points.** Displays the characters hit points.
- **Score.** Displays the players current score.
- **Help.** An interactive button which brings up a pop-up containing general help. The general help includes game rules, game goals, brief description of game mechanics including the functions and also constructs.
- **Level Tips.** An interactive button which brings up a pop-up that contains information to help to complete the current level.
- **Save.** A button that saves the players' current game state.

Chapter 5

Implementation

5.1 Technologies Used

The following subsections provide a high-level discussion of the technologies used during the implementation process. It aims to highlight and reflect the implementation approaches, any ancillary techniques used and the choice of programming language to develop the game while providing justification for such choices.

5.1.1 Source Control

Source control is a useful tool to help develop software. The source control system used for this study was GitHub. GitHub's source control system has several features that helped in the development process of the game. These are, the ability to save code locally and in their cloud servers, record and revisit code updates, and check merge changes. Code was committed every time a new functionality was added, or when there was a change to the code. This ensured that code was not committed too often, as this would be time consuming, and that the committed code was fully functioning.

5.1.2 Language Choice

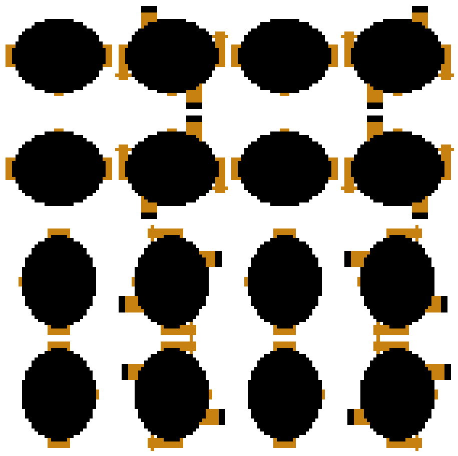
The game was developed using Node.js, a JavaScript runtime environment. The game was written in JavaScript, Jade and CSS, and is entirely client side; there is no game server. This allows quick execution of code as all execution occurs within the users' browser; time saved sending and receiving messages to and from a server.

JavaScript is a programming language for frontend development. The main reason for using it is because of its simplicity, vast functionality and because it is supported in many browsers. This includes the three most popular web browsers, Google Chrome, Safari and Firefox (StatCounter, 2019). JavaScript was used for the entirety of game logic, browser control, animating page elements and creating interactive content.

Jade was used for defining the web page and scripts. It is a markup language that is simpler than HTML and has more functionality. CSS is a style sheet language and was used for describing the presentation of elements, such as positioning, colour, size and more, within the web page.

5.1.3 Character Animations

To increase the entertainment aspect of the game a sprite was created for the character, enemy and bullet using Piskel, a free online sprite editor. A sprite is a two-dimensional image that contains multiple smaller images. These smaller images are used to create an animation.


(a) Sprite sheet for character

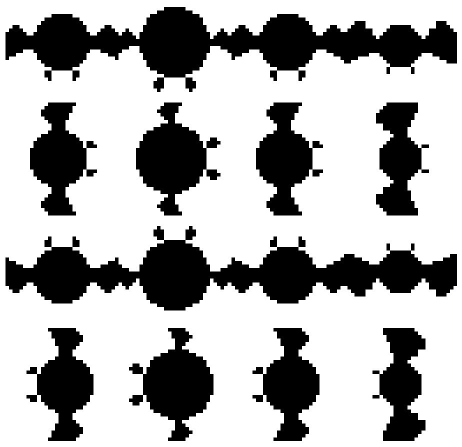

(b) Sprite sheet for enemy

Figure 5.1.1: Sprite sheets for character and enemy in Programmer's Dungeon

The JavaScript function, drawImage, was used to create the animation. drawImage takes parameters to specify which sections of an image to draw. This function was executed in a loop that updated the parameters with a counter, and thus creating the animation.

5.1.4 Hosting Game

Since one of the requirements was for the game to be accessible from home, work or school, it was necessary to host the game onto the world wide web. Anyone with a device with a browser connected to the internet would be able to access the game. This is not to mean that everyone who satisfies these requirements will play the game, but instead, users who wish to play, can do so from most locations.

There are many different hosting services that are specifically designed Node.js applications. However, I decided to use Amazon Web Services (AWS) as I had most familiarity with this software. AWS also provided a free domain name, with the URL: *http://tutorial-env.4yzpcwhdm6.us-west-2.elasticbeanstalk.com/*. However this URL is fairly long and does not reflect the contents of the web page in any way. So a more suitable URL was chosen through the service, freenom: *http://programmersdungeon.tk/*. This was set-up to do a URL redirect to the URL provided by Amazon, as the game is hosted there. This URL is free for a year and so will be usable throughout the duration of this study.

5.2 Overview of System Architecture

The following subsections provide an overview of the non-trivial components of the game. These are the map, character and console. The user interacts with the console to influence the characters actions, which in turn updates the game state and displays the changes on the map.

5.2.1 Map

The 'map' refers to level layout in the dungeon, specifically this is the walls represented by black lines, the enemies represented by bats, and the exit tile represented by a red square. The character is not a part of the map, but instead 'within' the map. The implementation of the map is comprised of two elements. One is what the user sees (see Figure 5.2.1), and the second is a co-ordinate system that the character interacts with. This co-ordinate system has X and Y axis, with origin (the 0,0 co-ordinate) in the top left corner of the map. This co-ordinate system is measured in pixels; one unit of length is one pixel. The map that the user sees is created using graphical lines, graphical rectangles and enemy sprite animations.

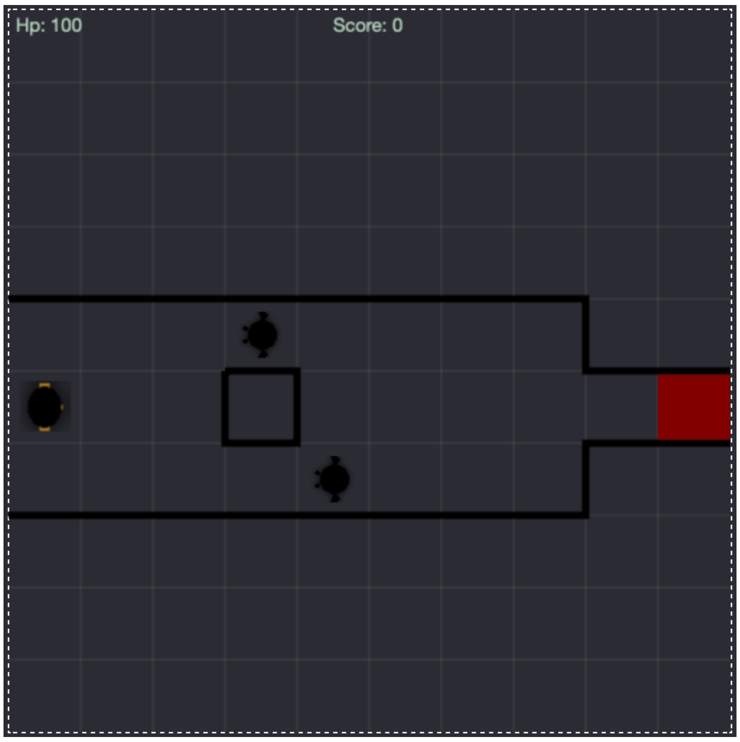


Figure 5.2.1: The map of level 6 in Programmer's Dungeon

The dimensions of the map are calculated, in pixels, using the *window.outerHeight* function. This means the map size is determined with regards to the user's browser window size. This allows the map to increase in size for larger screens and decrease in size for smaller screens. The graphical images and enemy size, conform to the size of the map as they take into account the dimensions of the map. This means that they remain in the correct location with appropriate size regardless of user screen size. Thus the game should look the same and be playable on different screen sizes.

The dark grey background and light grey lines in Figure 5.2.1, is present in every level. The light grey lines form the 'tiles' of the map. The character can only move from its current tile to any adjacent tiles, and there are ten tiles across and up the map. The distance the character moves is calculated by map.length/10. Since the map length and height are the same, this is used for the character to travel both horizontally and vertically.

5.2.2 Character

I have discussed how the character is animated and how it moves. Now I describe how it interacts with its environment, for example; getting hurt by an enemy or stopping against a wall.

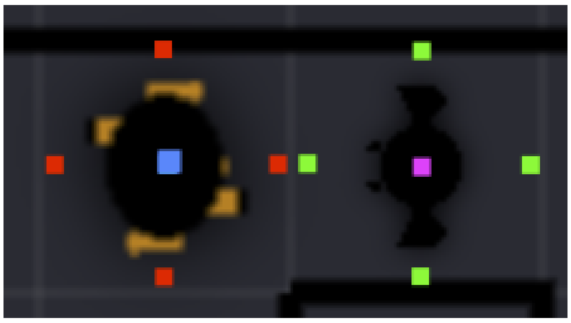


Figure 5.2.2: Contact points for character and enemy in Programmer's Dungeon

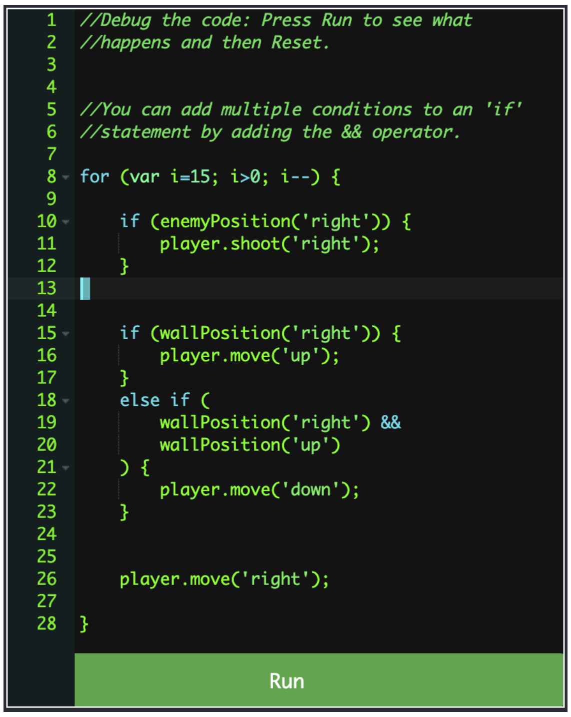
The character has X and Y co-ordinates that represent its centre, which is used to move the character across the map. However, for the character to interact with its environment, it needs a 'contact' point(s). This would be used to determine whether the character is 'touching' anything else. The contact point could have easily been the characters centre. However this was not a good idea since the character would have been able to get almost half its body across a wall before having to stop. This would not seem right. So the character was given four points of contact at its edges. In Figure 5.2.2, the blue square represents the centre of the character and the red squares represent its contact points. The contact points are derived by adding/subtracting a fixed pixel length to its centre in the X and Y direction. Enemies have the same set of characteristics.

This means the character would stop just before a wall, as walls run down the middle of two tiles. For enemies, since they also have a similar configuration of contact points, the character and enemy are deemed to be touching if any of the characters's contact points are between all four of the enemys. If they are touching, then the character loses hit points.

5.2.3 Map Walls

The walls which are represented as black lines, are just graphical images. For them to interact with and stop objects such as the character, enemy and bullets, they need to be represented in the co-ordinate system. Such that if any of the character's contact point ever touched the contact point of a line, it would have to stop moving in that direction. Instead of defining every pixel in the co-ordinate system where a wall is present, a 2D array was used to define a group of pixels as the former would be very tedious to do. This 2D array represents the whole map and is comprised of three elements. A 0 represents space that could be moved around in, 1 represents a wall, and 2 represents the exit. The character can move around in areas where there is a 0, is stopped from moving if in contact with a 1, and completes the level if in contact with a 2.

5.2.4 Console



```
//Debug the code: Press Run to see what
//happens and then Reset.


//You can add multiple conditions to an 'if'
//statement by adding the && operator.

for (var i=15; i>0; i--) {

    if (enemyPosition('right')) {
        player.shoot('right');
    }


    if (wallPosition('right')) {
        player.move('up');
    }
    else if (
        wallPosition('right') &&
        wallPosition('up')
    ) {
        player.move('down');
    }


    player.move('right');

}
```

Figure 5.2.3: The default console layout for level 6 in Programmer's Dungeon

The console was developed using Ace, an open-source code editor written in JavaScript (Ace, 2019), and a 'Run' button. The Ace functionality has a method which execute the code within the code editor. The button was implemented to call this method when clicked.

The level objectives are given through the comments at the top console. This is explained within the tutorial. Below the objectives, there are usually another set of comments. These are information about constructs. They either describe a new construct, or explain more about a construct that is being used or required in the current level.

Once run is clicked on the console, a complex system takes place to turn the code in the editor into actions carried out by the character. This would not be easy to explain without diving into many specifics, of which would not be of much help to this study. Instead, a high-level overview is that 1) all the code in the editor is executed before the character moves, 2) at the end of the execution, the character carries out all the pre-determined actions. Since the enemies do not move, this is no issue.

Chapter 6

Testing

Throughout development, the game was tested to ensure that it was working as expected. Once implementation was complete, three phases of testing were defined; system testing, obtaining initial feedback from user playthroughs and user testing. This chapter will discuss these testing methods in more detail, specifically why they were chosen and how they will be carried out.

6.1 System Testing

Testing was carried out throughout development. This was ensure that the game functionalities were working as expected and graphics were represented correctly. This covered the majority of unit test cases and integration test cases; every time a component was added, it was tested, and every time two components were combined, their input and outputs were tested.

Once implementation is complete, the final independent testing is to carry out system testing. System testing refers to testing the complete and integrated software. For this software this translates to playing through the entire game (or in other words, playthroughs).

To ensure that playthroughs were carried out in order to efficiently test the system, two objectives were determined. The aim of these objectives were to influence the playthroughs conducted. The objectives are:

1. Check if functional requirements are met.
2. Conduct certain edge cases were there is a high chance to discover bugs.

The first objective is clearly essential; the game was developed in order to teach iterative and conditional control structures by following the requirements and design. As the non-functional requirements require input from users in order to check that they were fulfilled, these could only be checked after user testing. With the second objective, since the game was developed independently, I had most familiarity with the software. Thus the focus was on areas of the game that were perhaps overlooked or had resulted in unexpected behaviours due to the implementation of other vital components.

6.2 Initial Feedback

Initial feedback from user playthroughs would be used in order to identify and consequently fix any obvious flaws in the game. Since the game was solely implemented by myself, any outside opinions would be very useful for numerous reasons.

The main objective of the feedback was to determine what users felt about the difficulty, usability and enjoyment of the game. As well as any other suggestions that they had. To conduct these tests, the think-aloud protocol was employed. This protocol is a method of usability testing. Usability testing refers to evaluating a product by testing it on users. It is widely used in the gaming industry and is very effective (Desurvire and El-Nasr, 2013). It involves players verbally expressing thoughts during gameplay. This might include what they are experiencing, their actions or feelings, and then collating this data.

To ensure feedback is as relevant as possible, participants with appropriate backgrounds and interests will be considered. In addition to the think-aloud process, the participants will be asked afterwards to provide suggestions for changes to the game in areas they felt needing improvement.

6.3 User Testing

The final phase of testing it to carry out user testing. The aim of this stage is to identify the learning, engagement and motivation of users through playing the game, and measure to which extent the non-functional requirements were fulfilled.

The design of the testing will be pretest-posttest. The testing will be comprised of three parts. Participants will first be given a pretest quesstionnaire, which will aim to gauge their interests, programming experience and some information about themselves. This should take roughly 10minutes to complete. They will then play through the game, without being forced to finish it. Finally they will be given a posttest questionnaire to identify the effects of the game in the domains of learning, engagement and motivation.

Chapter 7

Results

7.1 System Testing Results

All functional requirements were easy to check. Moreover, game functionality was based on these requirements, so these would have been checked during implementation. However to check these were working correctly, they were checked once more.

The key playthroughs that were conducted, were, playthroughs in the three most popular browsers; Safari, Google Chrome and Firefox.

7.2 Initial Feedback and Improvements

7.2.1 Feedback

Participant 1

The first participant was a first year undergraduate computer science student from Royal Holloway University. The following comments were gathered:

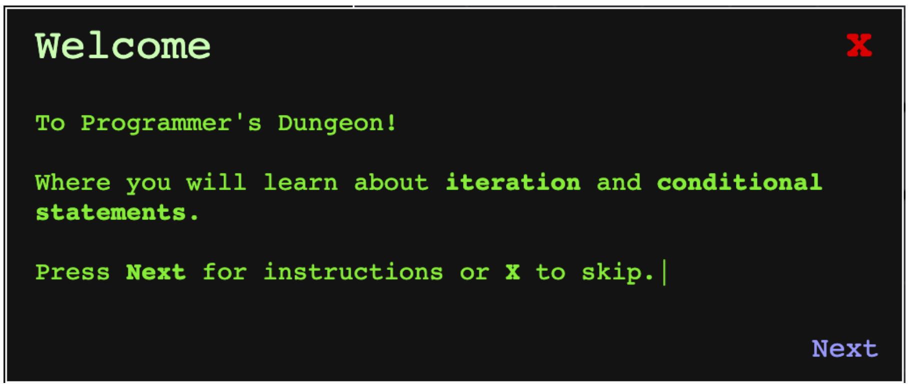
- Too many comments in the console. There is a lot of information to take in. No suggestion.
- Level 6, which requires debugging a more complex *for* and *if* code, was too difficult. Suggested making level easier.
- Level 7, which requires completing a nested *for* loop, too difficult. No suggestion.
- Last level, level 8, which requires creating your own code to a complex level, was most challenging and enjoyable level.

Participant 2

The second participant was an aerospace engineer graduate who was interested in learning how to code. The following comments were gathered:

- Too much text in the tutorial. Suggested making it more precise.
- Colour scheme of tutorial does not match colour scheme of the game. Suggested making tutorial match colour scheme of console.
- Reset level removes any changes to the code by resetting the text in the console back to default. Suggested not resetting the code, as you are not able to see your mistakes.
- Suggested scrolling animation for text in the tutorial text box.
- Help section is very useful because you can check construct and function definitions.

7.2.2 Changes


Welcome

To Programmer's Dungeon!

Where you will learn about **iteration** and **conditional statements**.

Press **Next** for instructions or **X** to skip.

Figure 7.2.1: The tutorial text box in Programmer's Dungeon

The feedback obtained was very helpful. It identified issues with level difficulty, game appearance and information provided throughout the game. Most changes were easy to implement as they only required changes to colour or text. Making the tutorial text scroll was a more complex task. This change was implemented because this made the tutorial more interesting and gave the game a more programming feel. The following changes were implemented:

- Tutorial text is more concise; only provides information to introduce the game. Some information was moved to the help section.
- Colour scheme of tutorial now matches the colour scheme of the console (see Figure 7.2.1).
- Tutorial text scrolls across the text box.
- Comments in the console about level information and construct information was thought over and made more concise.

- In level 6 and 7, constructs were kept the same, but the code was changed to make the level easier by making solutions more obvious.

These changes should result in making the game more aesthetically pleasing, making the difficulty curve smoother and not overwhelming users with too much information.

7.3 User Testing

7.3.1 Learning

7.3.2 Engagement

7.3.3 Motivation

Chapter 8

Discussion

Chapter 9

Future Work

Here, it is important that not only the final outcomes are assessed, but also that the learning and training process itself is monitored continuously without impairing the playing/learning experiences (e.g. via psycho physiological measurements or automated logs/recordings of player behaviour). This is especially beneficial as it can inform new ways to make learning games more adaptive so that they can always offer help or additional information when the players need it (e.g. when they get stuck at a certain point of a game) (Breuer and Bente, 2010).

Bibliography

AAStocks, 2016. Number of active video gamers worldwide from 2014 to 2021 (in millions) [Online]. Accessed: 2019-04-12. Available from: `https://www.statista.com/statistics/748044/number-video-gamers-world`.

Abt, C.C., 1970. *Serious games: The art and science of games that simulate life in industry, government and education*. New York, NY: Viking.

Ace, 2019. The high performance code editor for the web [Online]. Accessed: 2019-05-04. Available from: `https://ace.c9.io/`.

ACM-IEEE, 2013. Computer science curricula 2013 [Online]. Accessed: 2019-04-02. Available from: `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`.

AlliedMarketResearch, 2017. Serious games market [Online]. Accessed: 2019-04-04. Available from: `https://www.alliedmarketresearch.com/serious-games-market`.

Arkorful, V. and Abaidoo, N., 2015. The role of e-learning, advantages and disadvantages of its adoption in higher education. *International Journal of Instructional Technology and Distance Learning*, 12(1), pp.29–42.

Barnes, T., Richter, H., Powell, E., Chaffin, A. and Godwin, A., 2007. Game2learn: Building cs1 learning games for retention. *SIGCSE Bull.*, 39(3), pp.121–125.

Bennedsen, J. and Caspersen, M.E., 2007. Failure rates in introductory programming. *AcM SIGcSE Bulletin*, 39(2), pp.32–36.

Bisson, C. and Luckner, J., 1996. Fun in learning: The pedagogical role of fun in adventure education. *Journal of Experiential Education*, 19(2), pp.108–112.

Breuer, J. and Bente, G., 2010. Why so serious? on the relation of serious games and learning. *Journal for Computer Game Culture*, 4, pp.7–24.

Concannon, F., Flynn, A. and Campbell, M., 2005. What campus-based students think about the quality and benefits of e-learning. *British Journal of Educational Technology*, 36(3), pp.501–512.

Connolly, T.M., Boyle, E.A., MacArthur, E., Hainey, T. and Boyle, J.M., 2012. A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education*, 59(2), pp.661–686.

Cordova, D.I. and Lepper, M.R., 1996. Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of educational psychology.*, 88(4), pp.715–730.

Csikszentmihalyi, M., Abuhamdeh, S. and Nakamura, J., 2014. Flow. *Flow and the foundations of positive psychology.* Springer, pp.227–238.

Desurvire, H. and El-Nasr, M.S., 2013. Methods for game user research: studying player behavior to enhance game design. *IEEE computer graphics and applications*, 33(4), pp.82–87.

Djaouti, D., Alvarez, J., Jessel, J.P. and Rampnoux, O., 2011. *Origins of serious games*, Springer London, pp.25–43.

Ferguson, C.J., 2007. The good, the bad and the ugly: A meta-analytic review of positive and negative effects of violent video games. *Psychiatric Quarterly*, 78(4), pp.309–316.

Feurzeig, W. et al., 1969. Programming-languages as a conceptual framework for teaching mathematics. final report on the first fifteen months of the logo project.

Filsecker, M. and Hickey, D.T., 2014. A multilevel analysis of the effects of external rewards on elementary students' motivation, engagement and learning in an educational game. *Computers & Education*, 75, pp.136–148.

Gee, J.P., 2003. What video games have to teach us about learning and literacy. *Comput. Entertain.*, 1(1), pp.20–20.

Giannakos, M.N., 2013. Enjoy and learn with educational games: Examining factors affecting learning performance. *Computers & Education*, 68, pp.429–439.

Gidget, 2019. [Online]. Accessed: 2019-04-17. Available from: `https://www.helpgidget.org`.

Girard, C., Ecalle, J. and Magnan, A., 2013. Serious games as new educational tools: how effective are they? a meta-analysis of recent studies. *Journal of Computer Assisted Learning*, 29(3), pp.207–219.

Granic, I., Lobel, A. and Engels, R.C.M.E., 2014. The benefits of playing video games. *American psychologist.*, 69(1), pp.66–78.

Hayes, E.R. and Games, I.A., 2008. Making computer games and design thinking: A review of current software and strategies. *Games and Culture*, 3(3-4).

Koulouri, T., Lauria, S. and Macredie, R.D., 2014. Teaching introductory programming: A quantitative evaluation of different approaches. *Trans. Comput. Educ.*, 14(4), pp.26:1–26:28.

Lee, M.J. and Ko, A.J., 2011. Personifying programming tool feedback improves novice programmers' learning. *Proceedings of the seventh international workshop on computing education research*. pp.109–116.

Lee, M.J. and Ko, A.J., 2015. Comparing the effectiveness of online learning approaches on cs1 learning outcomes. *Proceedings of the eleventh annual international conference on international computing education research*. ACM, pp.237–246.

Lehrer, R., 1986. Logo as a strategy for developing thinking? *Educational Psychologist*, 21(1-2), pp.121–137.

Leppänen, M., 2007. A context-based enterprise ontology. *International conference on business information systems*. Springer, pp.273–286.

Malan, D.J. and Leitner, H.H., 2007. Scratch for budding computer scientists. *ACM Sigcse Bulletin*, 39(1), pp.223–227.

Malone, T. and Lepper, M., 1987. Making learning fun : a taxonomic model of intrinsic motivations for learning. *Conative and Affective Process Analysis*.

Michael, D.R. and Chen, S.L., 2005. *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade.

Moreno-Ger, P., Burgos, D., Martínez-Ortiz, I., Sierra, J.L. and Fernández-Manjón, B., 2008. Educational game design for online education. *Computers in Human Behavior*, 24(6), pp.2530–2540.

Muratet, M., 2012. Prog&play [Online]. Accessed: 2019-04-17. Available from: `http://progandplay.lip6.fr/index_en.php`.

Muratet, M., Torguet, P., Viallet, F. and Jessel, J.P., 2011. Experimental feedback on prog&play: a serious game for programming practice. *Computer graphics forum*. vol. 30, pp.61–73.

Newzoo, 2017. Distribution of video gamers worldwide in 2017, by age group and gender. [Online]. Accessed: 2019-04-12. Available from: `https://www.statista.com/statistics/722259/world-gamers-by-age-and-gender/`.

Newzoo, 2018. *2018 global games market report* [Online]. Available from: `https://resources.newzoo.com/hubfs/Reports/Newzoo_2018_Global_Games_Market_Report_Light.pdf?`

Prensky, M., 2001. Digital natives, digital immigrants part 1. *On the Horizon*, 9, pp.1–6.

Prensky, M., 2003. Digital game-based learning. *Computers in Entertainment (CIE)*, 1(1), pp.21–21.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J.S., Silverman, B. et al., 2009. Scratch: Programming for all. *Commun. Acm*, 52(11), pp.60–67.

Riccardi, G.A., 1981. The independence of control structures in abstract programming systems. *Journal of Computer and System Sciences*, 22(2), pp.107–143.

Ritterfeld, U. and Weber, R., 2006. Video games for entertainment and education. *Playing Video Games: Motives, Responses, and Consequences*, pp.399–413.

Robins, A., Rountree, J. and Rountree, N., 2003. Learning and teaching programming: A review and discussion. *Computer science education.*, 13(2), pp.137–172.

Roungas, B. and Dalpiaz, F., 2015. A model-driven framework for educational game design. *International conference on games and learning alliance*. Springer, pp.1–11.

Sangrà, A., Vlachopoulos, D. and Cabrera, N., 2012. Building an inclusive definition of e-learning: An approach to the conceptual framework. *The International Review of Research in Open and Distributed Learning*, 13(2), pp.145–159.

Scratch, 2019. Scratch statistics [Online]. Accessed: 2019-04-16. Available from: `https://scratch.mit.edu/statistics/`.

Sherry, J., Greenberg, B., Lucas, K. and Lachlan, K., 2006. Video game uses and gratifications as predictors of use and game preference. *International Journal of Sports Marketing and Sponsorship*, 8, pp.213–224.

Sicart, M., 2008. Defining game mechanics. *Game Studies*, 8(2).

StatCounter, 2019. Global market share held by the leading web browser versions as of february 2019) [Online]. Accessed: 2019-04-28. Available from: `https://www.statista.com/statistics/268299/most-popular-internet-browsers/`.

Susi, T., Johannesson, M. and Backlund, P., 2007. *Serious games : An overview.* University of Skövde, School of Humanities and Informatics.

UniversityOfBath, 2018. Department of computer science programme catalogue 2018/9 [Online]. Accessed: 2019-04-18. Available from: `http://www.bath.ac.uk/catalogues/2018-2019/cm/USCM-AFB06.html`.

Vygotsky, L.S., 1978. *Mind in society: The development of higher psychological processes.* Harvard university press.

Wang, H. and Sun, C.T., 2011. Game reward systems: Gaming experiences and social meanings. *Digra conference.* pp.1–15.

Watson, C. and Li, F.W., 2014. Failure rates in introductory programming revisited. *Proceedings of the 2014 conference on innovation & technology in computer science education.* ACM, pp.39–44.

Whitton, N., 2011. Game engagement theory and adult learning. *Simulation & Gaming*, 42(5), pp.596–609.

Wikiversity, 2019. Control structures [Online]. Accessed: 2019-04-18. Available from: `https://en.wikiversity.org/wiki/Control_structures`.

Wilkinson, P., 2016. *A brief history of serious games*, Springer, pp.17–41.

Wouters, P., Nimwegen, C. van, Oostendorp, H. van and Spek, E.D. van der, 2013. A meta-analysis of the cognitive and motivational effects of serious games. *Journal of Educational Psychology*, 105(2), pp.249–265.

Zhang, D., Zhao, J.L., Zhou, L. and Nunamaker, Jr., J.F., 2004. Can e-learning replace classroom learning? *Commun. ACM*, 47(5), pp.75–79.

Zyda, M., 2005. From visual simulation to virtual reality to games. *Computer*, 38(9), pp.25–32.

Appendices

Appendix A

Uncertainty Analysis

Appendix B

Screenshots

Appendix C

Ethics Checklist