Code Lab - A Game That Teaches High Level Programming Languages

Robert White, Feng Tian, Peter Smith
Faculty of Science & Technology, Bournemouth University, UK

Abstract- **With a sharp rise in the adoption of technology throughout the younger generation there is a sudden need for more technologically focused teaching methods in the educational sector. Many applications and websites offer games to students in order to decrease the learning curve associated with challenging subject matter. However, very few of these games have strived to teach high level programming languages to younger students. We propose in this paper Code Lab, a game that utilizes prominent learning theories, to structure the educational content, whilst using game design theory to attune the visual and mechanical design to appeal to students aged between 14 and 16. The evaluation results has shown that Code Lab helps students understand some basic programming concepts, though it is vital to balance between learning and entertainment through rewarding students and including less obstrusive learning material in the game.**

Keywords – Gamification; Game Based Learning; Game Design

I. INTRODUCTION

Technology has become a large part of the younger generations daily activities, with 90% of 16-24 year olds, in the UK, declaring that they own a smartphone [1]. This sharp increase in the adoption rate of technological advancements, in today's youth, has resulted in the educational system utilizing new methods of teaching via the usage of tablets and touchscreen devices. As a result, new innovative methods of teaching, such as the gamification of education, have emerged as a means of targeting this new demographic of tech savvy youths.

The concept of the gamification of education is simply stated as the teaching of a subject's content via the usage of video games and is still a fairly fresh and underdeveloped area. It is unclear just how long video games have been used in the classroom but it is commonly assumed to have arisen at around 2010. Through the use of games, it is hoped that students, who are being taught typically dry subjects in a manner that may not be engaging or fun for them, are shown new ways to learn that are more interesting. Ultimately video games could act as a middle ground between education and entertainment.

The gamification of education has already seen huge successes through such games as 'Minecraft'; the third bestselling videogame of all time (18.2m units) [2]. Minecraft has become such a staple of Games Based Learning (GBL) that Microsoft bought the franchise in 2014 for $2.5 billion and has decided to create and support a fully functional Educational Edition. At its current state the game is used for teaching anything from STEM subjects to art and poetry [3]. Such universal usage of a videogame in classrooms shows the power of games as a means for educating people.

In this paper we propose and develop an educational game, called Code Lab. It is a first person puzzle game aimed at young learners and is designed to be an interactive and engaging experience that, through the use of interesting game mechanics, teaches the player some simple high level programming concepts in ways that differ greatly to the current systems of teaching. In the rest of the paper we will present the rationale and details of the game design for Code Lab in Section II and III. The evaluation of the game will be given in Section IV, aiming to analyse the effectiveness of both the educational and entertainment aspects of the game, along with a conclusion in Section V.

II. WHY GAMES?

Typically, in an educational setting, students are taught subject matter via the use of various different media at separate times. One of the distinct advantages of video games is the fact that they utilize multiple types of media simultaneously and therefore benefit from the ability to stimulate both the visual and auditory channels of humans at the same time [4]. It is believed that through using this process to teach someone new information, they will remember significantly more than another person who is shown the same information but through the usage of just one type of media.

Video games have also been signified as an appropriate medium for teaching due to the interactive nature, the ability to discover new things [5] and the trial and error elements [6] that aren't as prevalent, nor even present, in some other forms of media. When considering the interactive implications of the medium, for educational purposes, it is clear that the use of video games enables the user to get real-time feedback from their inputs, which may not always be possible in conventional teaching situations. As such, a student can use the moment to moment feedback that a game provides to teach themselves, discover new links between subject matter and develop their understanding at their own pace [7].

Therefore, it is believed that the non-obtrusive nature of games and their place as pieces of entertainment should act as an enticing piece of media for prospective learners to engage with. Utilizing graphics, sounds, animations and text as core content, should ensure that Code Lab stimulates the

player's visual and auditory channels to an appropriate degree without being overly aggressive in its presentation.

III. GAME DESIGN

Code Lab was designed to be tested on a younger audience but doesn't limit itself to that specific age range and can therefore be used inside of an educational establishment or outside in a more casual environment. A number of learning theorists, game designers and emerging professionals work in the field of gamification have been researched to ascertain some of the key features and techniques that have been applied to the design of Code Lab.

A. The Game

Code Lab is a first person puzzle game that is designed to introduce the player to simple C# programming structure, syntax and variable types. The game achieves this goal through the use of two core game mechanics: programming jigsaws [8] and object hacking [9].

The programming jigsaws, shown in Fig. 1, act as the introduction puzzles to the game and involve the player being presented with a programming example that needs to be solved. By selecting blocks from a pool of predefined code pieces the player begins placing them into a slot on the blank problem. Once the player thinks that their answer is correct, they submit their answer by interacting with the 'Run' button.

Figure 1. Image of first string variable jigsaw puzzle

A jigsaw type puzzle was used as the first problem type for the player to solve because it shares similarities with a visual programming interface [8] that the user may have experienced prior to this game but with the difference of using actual real world programming syntax [7].

The second puzzle type, object hacking, shown in Fig. 2, occurs when the player interacts with an object in the environment that displays the {} symbol when it is being looked at. Hacking an object results in a coding UI appearing that shows a number of pieces of information to the user. The left hand side of the screen shows the current object's attributes, which can be edited, and the object's methods which can be used in the code section. To the right of the screen is the current coding problem associated with the object being hacked. The problem located on the right must be solved in the code section in the middle of the screen and must be written in C#.

Figure 2. Image of first *if* statement hacking room puzzle

When the player feels like their input code is correct and will solve the object's problem, they press the 'Run' button and it will check the code to see if the problem has been solved. If the answer is correct, the object's variables will become manipulatable and will affect the object's properties inside of the game world. For example, there is a platform object in the game that can move up and down. Once the platform object has been successfully hacked, the player can change the variable 'direction' to be either "up" or "down". When the platform's 'Run' button is pressed, this will result in the platform moving either "up" or "down" in the game world. The hacked objects act as sub puzzles and are used by the player to progress through the room that they are currently inside of.

The player progresses through increasingly complex puzzle rooms that introduce new objects to hack with unique and interesting properties until they reach the end of the testing session and complete the game

B. Real Code, Real Syntax, Real Structure

One of the most important features of the game is the use of real world programming syntax and structure. Many of the applications and coding websites available to younger learners operate using visual programming languages such as Scratch [8] and Code Academy [10]. Code Lab teaches the C# programming language and focuses on basic variable types, overall code structure and syntax through the use of *if* statements.

It was important to use a real world programming language for a number of reasons. Research into learning theorists have shown that people are more inclined to learn and pursue further knowledge when they are learning skills with clear real world applications [11]. This sentiment is further backed up

by [12] who claims that as humans we gravitate towards games with mechanics that are prevalently sought after in modern society.

C. Feedback, Signposting and Tutorialization

Research into game design and learning theories highlighted the importance of feedback, signposting and tutorialization in the presentation of the games content.

Code Lab begins with short tutorial puzzles that introduce the player to some of the basic C# variable types that they will encounter in the game. The opening tutorial is designed to be relatable to the player as a means of solidifying the information in their minds. This was achieved by linking the variables back to a human's personal information. For example, the game asks the player to assign an age to the 'int age' variable. These tutorials act as a means of solidifying the most basic forms of complex programming ideas into the user's mind [13] so that they can build upon the knowledge they have gained and use it to solve more complex problems [14] later on in the game [12] [6].

Tutorialization follows on throughout the game's levels via the use of tooltips, as seen in Fig. 3, providing the player information ranging from the interactions, that can be made in the environment, to assistance with programming. This type of tutorialization can be seen in many modern games and acts as a way of introducing the player to game mechanics when and as they are needed [15].

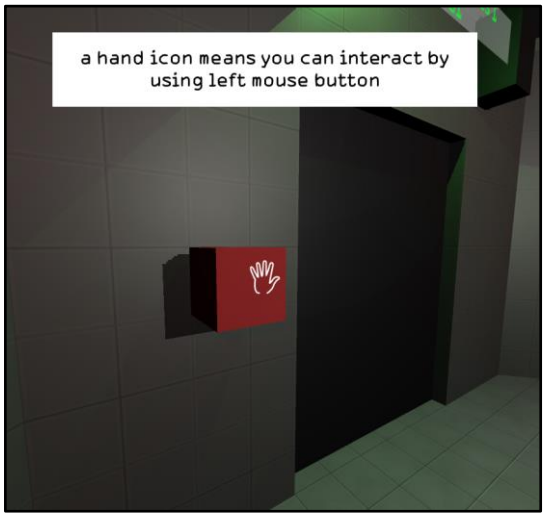


Figure 3. Image showing tutorial tooltips and mouse cursor changes

Another important aspect of the game's design was the feedback supplied to the player when they enter an incorrect answer. The game needs to positively assure the player and offer real and reliable feedback so that the user knows exactly what they have done wrong. As such, when the player runs their inputs for the jigsaw puzzles, the incorrectly input code blocks light up red and the correct blocks light up green; as shown in Fig. 1. This simple feedback loop allows the player to understand where they went wrong on these puzzles.

D. Limitations

Despite the number of good features mention above, the game does have some constraints.

A custom compiler would have been desired for the game so that the player could enter actual codes and have them compiled. The desire for this came from the game 'Else Heart.Break()' [9]. It gives the game more depth and a greater sense of realism if the player could write their own codes and get them compiled. On the other hand, it could be argued that the current state of the game is more casual and fitting for a newcomer to programming and as such is sufficient enough for the overall needs of the game.

There are twelve puzzles present in the current version and all focus on the introduction of variable types, methods and increasing difficulty of *if* statements in C#. Twelve puzzles did not seem enough for the game to introduce various different programming concepts along the way. Therefore, it seemed apt that the game should only focus on different variable types, methods, the usage of *if* statements and the conversion of binary numbers to decimal.

IV. EVALUATION

The game Code Lab was evaluated as an educational game for young students aged between 14 and 16 years old. Code Lab contained puzzles that included information regarding variable types, syntax and structure of *if* statements in the high level programming language C#. Students enrolled on Computer Science, Computing and Information & Communications Technology (ICT) General Certificate of Secondary Education (GCSE) courses from various different secondary schools across the United Kingdom were presented this game as a learning tool for use in tandem with their studies. Due to the difference in schools and courses, participants have different degrees of experience using high level programming languages which offers a good variation in testers.

A. Methodology

It was vital that, in order to evaluate the effects of the game, the participants completed a number of questionnaires throughout the duration of the process. A pre-game questionnaire was implemented into the game, focusing on obtaining some of the user's basic information such as their name, age, exposure to different programming languages and frequency of time spent playing games.

This information was vital for garnering an understanding of the differences between the participants and how their

experience in the game differed from others based on factors such as pre-existing programming knowledge.

A post-questionnaire was also factored in and asked the participant to offer feedback based on how much they enjoyed the game, things that they liked and disliked about the game, their opinion on the game as a learning tool and what they might improve or even remove from the game. This information was important because it allowed for an analysis of how well the game bridged the gap between education and entertainment and enabled the tester to offer feedback that could potentially improve the state of the game as a learning tool.

In addition to the questionnaires, it was vital that the participants were tested before and after playing the game so that a clear analysis of the change in their understanding of the game's subject matter could be attained. As such, one short and pre-game multiple choice test of 10 question was designed for the users that covered much of the learning content that the game hoped to teach. This first test did not reveal the correct answers to the player. Once the game had been completed, or when the player decided to stop playing the same test was presented to the user covering the same content as the first test but did not hide the answers. By not revealing the correct answers from the first quiz it was possible for us to avoid the participant from remembering their answers from the pre-test and just copying them for the post-test.

It was also important that the test results accurately represented the participants understanding of the C# programming language. Due to the multiple choice nature of the questions, it was decided that a large number of options should be offered to the tester in order to avoid them just guessing the answer correctly. This addition to the tests was also backed up by the option for the participant to signify that they did not know the answer to the question.

The game also collected analytics data from each participant that tracked some important factors. The data included how long it took for the player to complete the puzzles, how many times the player succeeded or failed to hack an object, how many times the player succeeded or failed a puzzle, the overall gameplay time and many other factors that were deemed interesting and vital to the overall analysis of the experience.

Participants were given 7 days to complete as much of the game as they wanted to as long as they filled out the pre and post-game questionnaires and tests and offered reasoning for why they did not want to finish the game so that their feedback could be analysed appropriately.

B. Pre-Game Questionnaire Results

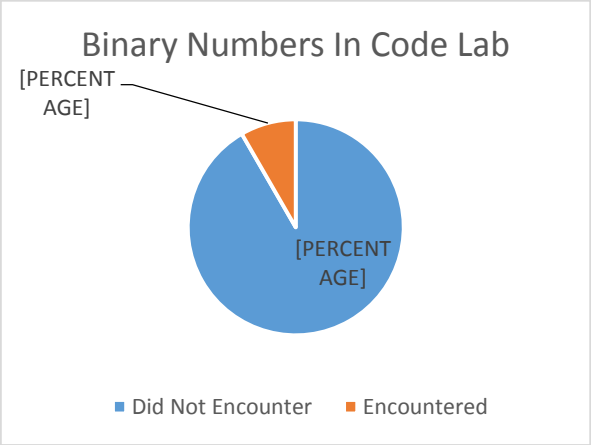
The final sample of participants consisted of 24 students at the GCSE level of education in the United Kingdom from schools in the Bournemouth area. Participants studied a myriad of subjects but all partook in at least one ICT, Computing or Computer Science related course. All of the participants signified that they had less than 3 months of experience with the C# programming language with 79.17% stating that they had never used the language before. The majority of participants, 87.5%, had encountered a different programming language before with 37.5% having used more than one other programming language. Of the programming languages mentioned the most used was Python with 62.5% followed by JavaScript 41.67%. Other programming languages included; Java, Scratch, HTML/CSS, Pascal and Visual Basic.

When asked how often they played video games, 50% of the participants stated that they played 10 or more hours a week, 41.67% played between 6-9 hours a week and only 8.33% opted for not spending any of their spare time playing video games. Of all the participants who took part in the study, only 12.5% could name any learning games. Despite not being a learning game, Scratch was mentioned by all of the participants who could name a learning resource. Minecraft was also mentioned by the participants.

C. Post-Game Questionnaire Results

Once the game had been completed, or the player stopped, they were asked questions about their experience. The participants were asked to grade how well they thought the game had developed their understanding of variables, *if* statements, binary and programming syntax by choosing one of 4 options ranging from “A lot” to “I did not encounter different variable types / *if* statements / binary / programming syntax whilst playing the game”.

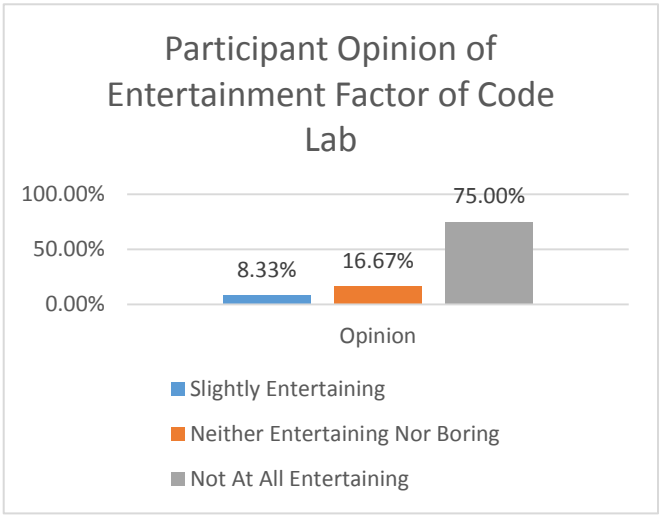
On the subject of variables, 41.67% stated that they felt like they had learnt “A lot” about the different variable types highlighted in the game, 20.83% of players felt like they had learnt “A little” and 37.5% did not learn anything additional about the different variable types.



When asked about their experience with *if* statements in the game, 16.67% stated that they had learnt "A lot", 45.83% learnt "A little", 12.5% did not learn anything additional to their previous knowledge and 25% did not encounter *if* statement related content featured in the game's later stages.

When asked about the development of their knowledge of binary numbers the majority of players, 91.67%, said that they did not encounter binary numbers at all throughout their time playing the game. Only 8.33% of players felt like their knowledge of binary numbers had improved "A little".

The question regarding improvements in understanding of programming syntax saw the highest approval rating. 83.33% stated that they had learnt "A little" about C# programming syntax whereas 16.67% felt like they had learnt "A lot" about it.



Participants were then asked about how entertaining they felt the game was. The testers were given 4 options ranging from "Very entertaining" to "Not at all entertaining". Of all the participants, 8.33% found the game to be "Slightly entertaining", 16.67% thought the game was "Neither entertaining nor boring" and, the majority, 75% felt that the game was "Not at all entertaining".

Following on from this question, the participants were simply asked whether they thought the game focussed too much on pushing the learning content, to which all of the testers stated "Yes".

The participants were then asked to comment on whether there were any features in the game that hampered their ability to learn from the games content. The individuals who did not spend any time playing video games in their spare time, stated that they initially had problems navigating the character around the environment, largely due to thinking that the arrow keys would be the movement controls, however this game used typical W, A, S, D PC key bindings for movement.
Testers were then asked to take notes about the features that they did and did not enjoy as well as anything that they would add to the game. Many of the participants liked the addition of the simplistic square characters in the game as they were funny to look at. The testers who were familiar with Scratch or other visual programming languages found the jigsaw puzzles to be easy and fairly familiar. Participants also highlighted the game music as entertaining to listen to.

Many of the participants stated that they thought that there was too much text present in the game and that as a result, they barely read it. It was also brought up that completing a puzzle did not feel that rewarding. Additionally, a few testers mentioned that the environment was very bland and square; they would have liked more varied rooms so that the puzzle locations did not look repetitive.

D. Analytics Data

It was estimated that the game would take 15 minutes to complete. The longest somebody spent playing the game in one sitting was 29 minutes and the shortest was 6. On average, the playtime was 14 minutes which includes 6 testers who did not finish the game and 18 who did.

An average of 2 minutes was spent by the players in order to complete the first puzzle; with the lowest time being 39 seconds and the longest being 3 minutes and 9 seconds.

The time taken to complete further jigsaw puzzles dropped dramatically to an average of 56 seconds between finishing the previous puzzle and finishing the next jigsaw type puzzle.

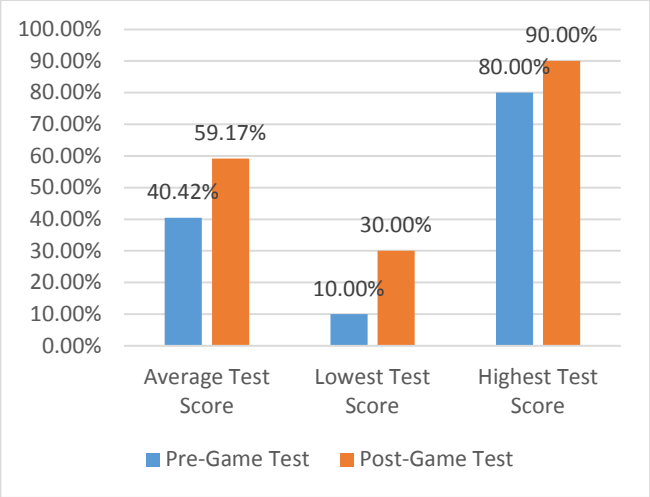
It took an average of 2 minutes and 45 seconds for players to complete the second puzzle, the first hacking puzzle, with the lowest time being 2 minutes and 14 seconds and the highest being 3 minutes and 20 seconds. The time taken to complete further hacking related puzzles also saw a significant drop in completion rate between the end of one puzzle and the completion of the next; showing an average completion time of 46 seconds.

Other analytics data recorded was how many times an individual failed to complete a jigsaw block or hacking puzzle. None of the participants failed any attempts to complete the jigsaw type puzzles. The most common failures, with regards to the hacking puzzle rooms, were; the first hacking puzzle room, the float hacking puzzle room and the final *if* statement room puzzle. The first hacking puzzle room had an average fail rate of 11 attempts. The float hacking puzzle room had an average fail rate of 5 attempts and the final *if* statement room puzzle had an average fail rate of 13 attempts. The boolean, integer and methods hacking puzzle rooms had no failed attempts.
The game also tracked how often the player backtracked into previous puzzle rooms. Participants, on average, moved back to the first string jigsaw puzzle room 3 times. There was no significant backtracking for the integer or boolean rooms but on average people did move back to the float and method jigsaw rooms 2 times. Most backtracking occurred in the final

stages of the game during the *if* statements puzzles whereby on average people backtracked to the *if* statement jigsaw room 4 times.

E. Learning Outcome

In order to assess the successes of the transferal of knowledge to the player a pre-game and post-game tests were taken by the participants.



The average pre-game quiz score was 40.42% with the lowest score being 10% and the highest being 80%. Participants performed better in the post-game quiz which saw an average score of 59.17% with the lowest score being 30% and the highest being 90%. Participants who had experience in more than one programming language performed much better than those who had only experienced one programming language. Those who had experience in high level programming languages performed better than those who had only used visual programming languages before.

Questions 1 and 4 were identified as the hardest questions for individuals to answer. Question 1 required the participant to select which answer could be assigned to a string variable. The correct answer was "None of the previous" as none of the other answers had the string of characters enclosed inside of a pair of quotation marks. In the pre-game quiz nobody got Question 1 correct but in the post-game quiz 29.17% of people got the answer correct.

The pre-game results for this question highlight a lack of knowledge in string variables as this was also the subject on which the majority of players spent time in the games puzzle rooms.
Question 4 asked the participant to select the answer that correctly instantiated a float variable. 16.67% of testers answered the question correctly in the pre-game quiz. 45.83% of participants selected the answer 4 which was incorrect because it did not contain an '*f*' at the end of the variable declaration. In the post-game quiz 29.17% of participants answered the question correctly. Much like the string variables, players made more errors in the float orientated puzzle rooms which signify a need for further understanding in these areas. Participants seemed to find variable types more difficult when they required additional characters to define their values, e.g. strings requiring double quotation marks, floats requiring an '*f*' suffix whereas integers merely require a number and booleans require either true or false.

The easiest questions to answer were Questions 2 (describing an integer), 3 (assigning a boolean), 8 (declaring the output of some code) and 9 (selecting the appropriate relational operator) which all saw above a 60% success rate in the pre-game test and then above 75% success rate in the post-game test. Question 5, on the subject of selecting the correct missing piece of syntax, saw the biggest increase in successful answers from the pre-game test average of 41.67% to 100% in the post-game test

Those who completed the game performed better in the post-game test than those who did not complete all of the puzzles. Participants whose scores were very high (70% or above) in the pre-game test tended to benefit less from the experience and on average only answered 1 more question correctly whereas those who scored very low in the pre-game test (30% or below) tended to answer 2 or 3 more questions correctly.
Question 7 asked the player to convert the binary number 0110 to decimal. There was no change in the number of people who answered this question correctly between the pre and post-game tests. It can therefore be deduced that those who answered this question correctly already had prior knowledge of binary and as such, none of the participants garnered any knowledge of binary numbers from this game.

V. CONCLUSION

Overall the results for this study were mixed. There was clear evidence to support the fact that the game did indeed develop the user's understanding of some of the basic C# programming concepts, so the educational aspects of the game were sound. However, there were a severe lack of elements that rewarded the participants for playing the game and as such, the entertainment factor was not that strong.

Once the participants had understood how the mechanics of the game functioned they began to work their way through the puzzles much easier and quicker.

Therefore, the tutorials were sufficient enough to teach the player the game's mechanics. On the other hand, it is worth noting that the players who did not identify themselves as playing video games in their spare time had a harder time understanding the basic controls of the game. In which case, further tooltips, popup tutorials and keybindings menus could be implemented in future builds. It seems appropriate that the tutorialization of the game should be designed with non-gamers in mind and to act under the assumption that the player may never have touched a keyboard before when teaching controls.

All of the participants thought that the educational aspects of the game were too much. This highlights a need for less obtrusive learning material in educational games. It appears that, to make an educational game that has a fine balance between entertainment and learning, the educational aspects have to be tied to the game mechanics but not in an obvious way. This would allow for a game that focusses on all of the important game design factors that make a game entertaining whilst tying information to the game mechanics for the player to remember whilst they are playing.

This study shows that this strategy of masking the learning material over already functional and entertaining mechanics could lead to greater enjoyment and, as such, a better adoption of knowledge on a subject.

Future study into this area would involve developing a new game that incorporates all of the feedback garnered from this experiment. We would look to develop a game that has very familiar, tried and tested gameplay mechanics and link some of the items, characters, environments into the learning material so that all of the language used in the game is reiterating core concepts of the target learning material.

VI. REFERENCES

[1] Statistics Portal. "Smartphone Ownership By Age 2012-2015 | UK Statistics". *Statista*. N.p., 2016. Web. 20 Apr. 2016.

[2] Lofgren, Krista. "2015 Video Game Statistics & Trends Who'S Playing What & Why? | Big Fish Blog". *Big Fish Games*. N.p., 2016. Web. 20 Apr. 2016.

[3] Microsoft. "Minecraft: Education Edition - Home". *Minecraft Education Edition*. N.p., 2016. Web. 20 Apr. 2016

[4] Mayer, Richard E. *Multimedia Learning*. Cambridge: Cambridge University Press, 2001. Print.

[5] Kulman, Dr. Randy. "5 Ways Video Games Teach Children Problem Solving Skills - Learningworks For Kids". *LearningWorks for Kids*. N.p., 2012. Web. 20 Apr. 2016.

[6] Bruner, Jerome S. *The Process Of Education*. Print.

[7] Saines, George. *Codecombat*. Silicon Valley, CA, USA.: CodeCombat, 2013. Print.

[8] *SCRATCH*. Cambridge, Massachusetts, United States: MIT Media Lab, 2013. Print.

[9] Svedang, Erik. *Else Heart.Break()*. Uppsala, Sweden: Erik Svedang AB, 2016. Print.

[10] Sims, Zach and Ryan Bubinski. "Learn To Code". *Codecademy*. N.p., 2011. Web. 20 Apr. 2016.

[11] Brown, John Seely, Allan Collins, and Paul Duguid. *Situated Cognition And The Culture Of Learning*. Palo Alto, Calif.: Institute for Research on Learning, 1988. Print.

[12] Koster, Raph. *A Theory Of Fun For Game Design*. Scottsdale, AZ: Paraglyph Press, 2005. Print.

[13] Papert, Seymor. "Constructivist Theories". *Mennta.hi.is*. N.p., 2016. Web. 20 Apr. 2016.

[14] Romiszowski, A. J. *Designing Instructional Systems*. London: Kogan Page, 1981. Print.

[15] Wawro, Alex. "The Art Of The Tutorial: When To Hold A Player's Hand, When To Let It Go". *Gamasutra.com*. N.p., 2016. Web. 20 Apr. 2016.