RUNJUMPCODE: AN EDUCATIONAL GAME FOR EDUCATING PROGRAMMING

Matthew Hinds, Nilufar Baghaei, Pedrito Ragon, Jonathon Lambert, Tharindu Rajakaruna, Travers Houghton and Simon Dacey
Unitec Institute of Technology, Private Bag 92025, Victoria Street West, Auckland 1142, New Zealand

ABSTRACT

Programming promotes critical thinking, problem solving and analytic skills through creating solutions that can solve everyday problems. However, learning programming can be a daunting experience for a lot of students. *RunJumpCode* is an educational 2D platformer video game, designed and developed in Unity, to teach players the fundamental concepts of C# programming. The game enhances the player's programming knowledge by providing a fun range of challenges and puzzles to solve. We promoted the interaction of programming through a 'Code Box', allowing players to enter lines of predefined code that modifies in-game objects. This tool is essential in completing the challenges and puzzles we designed. To allow alterations of its properties, we made further manipulation of each object possible, which would give the player creative freedom to complete each level. Quizzes and journals were utilized to assess and collate their learnt material for future reference. In addition, we created a mobile application to track each player's statistics throughout the game and compare their progress with other users. A full evaluation study has been planned, the goal of which is to examine the effect of using the system on students' learning.

KEYWORDS

Game design, education, programming.

1. INTRODUCTION

Programming promotes critical thinking, problem solving and analytic skills through creating solutions that can solve everyday problems. Technology in the 21st century has evolved in many ways, helping us to be connected. Through each of these innovations comes a need to develop and design new solutions. Getting into programming can be a daunting experience and has shown to be too overly complex, frustrating and unenjoyable for many novice students (Lahtinen, Ala-Mutka, and Jarvinen, 2005; Milne and Rowe, 2002).

Current easy to access resources consist of articles and written tutorials. There has also been attempts in improving teaching through classes, video and auditorial – being able to learn through a fun interaction can be a more enjoyable and knowledge retention experience. Video games are not a new concept and while they are popular and generally created for entertainment purposes, they can be adapted and used for engaging users with educational materials.

Determining the age group in designing educational programming video games comes from identifying the curriculum of which is planned to be taught. The more complex the content, the higher the recommended age bracket is. However, current video games that teach programming generally cater towards the ages 16 and below. This can be attributed to the fact that more complex and involved topics require a dedicated classroom teaching environment that offers live guidance through a teacher. Current educational video games, e.g. Code Spell (https://codespells.org/), Kodu (www.kodugamelab.com/) and Scratch (https://scratch.mit.edu/) typically teach the basic fundamentals and later on expand each topic resulting in a progressively increasing learning curve of content difficulty, which in turn can reduce the overall retention of players and learning speed.

While progressing the content naturally increases its difficulty, it is important to present the new material in a way that is easily accessible and personalise it towards the knowledge of the player. In this project, we slowed down the progression of new information and packaged it together with new gameplay mechanics so that players learn through naturally completing each level. In addition, after every three levels we included a quiz that recaps the previous levels' content. By doing so, players are reminded to pay more attention throughout each level so they can successfully pass the quiz and proceed to the next level.

Video games first emerged since the successful proof of concept by Physicist William Higinbotham in October 1958 using circuitry revolving around the use of resistors, relays, capacitors and transistors ("The First Video Game of William Higinbotham," n.d.). Since then during the next decade, video games have seen a radical change as general computing became more developed and commercially viable.

With the release of the first video game console, the Magnavox Odyssey the video game industry saw its first leap towards public interest and the industry began its capitalization of the at home entertainment system (Schilling, 2003; Williams, n.d.). Popularity with video games made its first big break with the release of the Atari 2600 in 1977 (Bellemare, Veness, and Bowling, 2012) and the Nintendo 64 in 1996 (Schilling, 2003); since then video games have caught the attention of millions of children and families which has strengthen its status as a must have electronic of the modern household.

Video game interaction revolves around user input on a touchscreen, gamepad or keyboard and mouse to control and perform the appropriated directed action desired on screen. These repetitious actions are fundamental in playing and through studies have shown to help improve motor functionality and hand eye coordination. The video game experience in particular allows gamers to develop perceptual and cognitive skills in many aspects that exceeds those of their non-gamer counterparts (Green and Bavelier, 2004).

The educational games genre has yet to see their popularity boom, as most players tend to use video games as a form of entertainment rather than education. However, in more recent times the use of video games to develop educational tools has expanded due to the rise of affordable and accessible technology, especially in smart tablet devices. It allows them to be an effective classroom tool to help students learn and reinforce a variety of skills and knowledge (P. Rossing, Miller, Cecil, and Stamper, 2012).

Rewards should be given out to players for completing specific tasks that range from easy to hard. The drive to seek out higher rewards come from video games tendency to promote competitiveness amongst players and their peers. Such behavior can be exploited to promote higher engagement and retention rates as players are more likely to work to completing more difficult tasks if they are tempted with a greater reward for completion. Such rewards are built around how the game is developed, but should be meaningful so that they do not feel worthless. For example, giving new unlocks for levels, badges, medals, titles or character customization options are suitable rewards and can garnish players' attention.

An educational game designer should first thoroughly understand the contents and methodologies of its subject. Without a strong and accurate background, it would be difficult to have confidence in the teaching material and curriculum. One can then decide on the best way to represent that content in a meaningful way. Each dimensional environment (2D or 3D) has its own strengths and weaknesses, one being more immersive through freedom of a three-dimensional world, but heavier on hardware taxing reducing the potential adoption of low end system players. While the other is less engaging through a lack of connection that is brought in from a two-dimensional world, it also allows a wider adoption due to less hardware constraints.

Following a teaching structure that is similar to a classroom curriculum will help build a path of content that is easy to grasp in the beginning and later becoming more difficult. Progressively expanding the content allows players to start off with the basics and slowly move up to more in-depth and complex concepts, giving them a natural curve of increasing knowledge.

2. RUNJUMPCODE GAME DESIGN

RunJumpCode is a 2D platform with basic left, right jumps & gravity idea. It was adopted from Super Mario Bros. (Pedersen, Togelius, and Yannakakis, n.d.) due to its high success rate (Ryan, 2011, Baghaei 2016), simplicity and level of entertainment. User have to learn theory, solve problems and apply learned knowledge in order to progress through the game.

Once launched, the player is taken to the main menu where items are clear and easy to understand. The idea behind the simplicity is so that the players on a broad age group will be able to figure out the workings of the GUI with minimal effort. And since this is the user's first point of contact with game and as a first impression, it was imperative for the design to be cluster free and easily understandable.

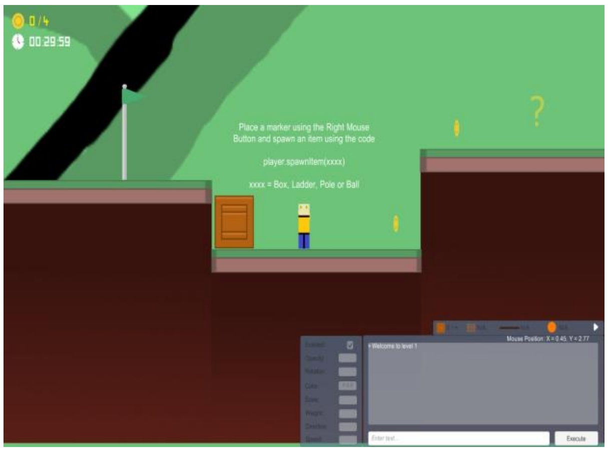
RunJumpCode was developed with a diverse age group in mind, mainly senior high school students and/or undergraduate tertiary students. The first step is to login which is done either with a new account or an existing account. Once logged in, the player will be taken to the level selection window where the completed levels and the next level are highlighted. Once clicked on a highlighted it will show the level name, description and the difficulty to give the user an idea as to what they are about to do and learn in order to complete the level. Once the user selects a level and click play, the level will be initiated. The first thing the user sees on the level is the level information screen. Key information shown here is the programming tutorial section, which highlights what is taught and the exclusions section, which highlights the restrictions applied to the level. This helps the users to familiarize themselves with the level so they know what to expect for the coming session.

The game character moves by detecting arrow key presses. The user has to navigate the game space to reach the endpoint by clearing obstacles along the way. Each obstacle poses a learning curve to the user. This is an approach to promote logical thinking. Logical thinking is by far the most important aspect of software engineering and this is where some people struggle. With our application, the users are actively encouraged to come up with solutions to each problem. To make things more interesting, each level also comes with a set of restrictions just like lifelike scenarios where not all the options are available. On each level these restrictions are different, to ensure the user cannot interact with a problem same way if it occurs multiple times, thus forcing them to think differently.

Obstacles also require object manipulation, meaning the user may have to spawn, scale, enable & disable objects to progress. This approach was encouraged by need to teach syntax. As an example to spawn a box a player has to enter "*player.spawnItem(Box);"*. If the user makes an error a syntax error will be shown with the cause, helping the user to understand how important it is to follow syntax in coding.

Apart from that, the user also has a theory section as well, giving the user key theory knowledge required to progress with the game such as explaining what Integers are in Level 1. This is further elaborated with examples which in our past experience helped with memorizing theory. Before the user can complete any level, there is a minimum number of coins to be collected, which is another measure taken to make sure the player gets a minimum level of knowledge before progressing forward.

The game starts at a very simple level, which is made even easier with hints and lots of mouse clicks and less coding. As the game progresses, it involves more and more coding and less GUI operations. This is also a systematic approach since people who are not very technical, seem to need more help till they get more familiar with the concept of coding. Although hints are available, it is not free to use them. The user loses some marks for using hints. This helps to discourage the player from using them and guides them more towards thinking and independent learning. Both of which are again quite important for software developers.



Place a marker using the Right Mouse Button and spawn an item using the code

player.spawnItem(xxxx)

xxxx = Box, Ladder, Pole or Ball

```
• Welcome to level 1
```

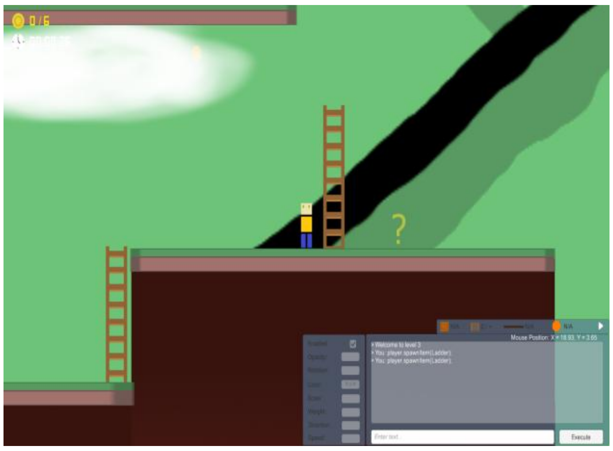
Figure 1. *RunJumpCode* Level 1.

In level 1 (shown in Figure 1) the obstacles are very simple to overcome. All the player needs to do is to place a box, jump on it and continue on. As the player progresses, the obstacles get progressively trickier. In figure 2 one box is not allowed so the user has to summon a ladder. Merely summoning one ladder is not enough as the player also has to scale the ladder. This makes the user interact more with the game on a creative level. With development tools such as Unity[1] and Unreal Engine[2] being available for free, 3D development has become slightly easier. Having said that, budget plays a role on how a game is being developed. 2D game development cost less as teams can be smaller in size and development is less complex. Math and art are more complicated in 3D. As for gameplay, 2D game has simpler physics mechanics and basic intuitive controls. Players would be able to identify interactive objects in the environment easily. Interactive tutorials and Code Box also work well on 2D because of the said simpler layout and should cater well to users of all ages compared to 3D. 2D development requires low system requirements thus the game can be played on a very low end system without sacrificing the gameplay. Another issue with 3D development is having to use more memory and resources thus producing inconsistent framerate could greatly impact player's experience on a slow machine.

Building a good programming foundation is one of the goal of this game. It aims to enhance player's knowledge by solving a wide range of challenges and puzzles as they go through each stage. The primary focus was the introduction of programming in a way that is mentally stimulating while being engaging and fun. This would provide the user an engaging experience and help increase knowledge of programming fundamentals.

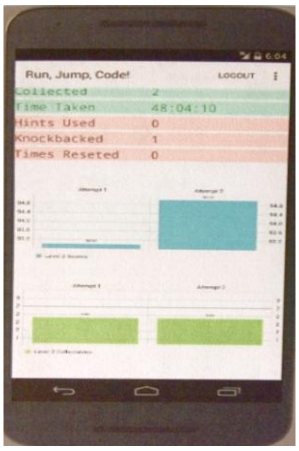
3. TARGET AUDIENCE

The game is designed to cater for teenagers and young adults. It is easy to understand and would be attractive to Senior High School students and young professionals. It can feed their interest with both gaming and programming. Those who are new to gaming should find an easy take on into the game. *RunJumpCode* is also challenging enough for regular game players and we believe parents might be interested in trying it out too.



```
> Welcome to level 3
> You: player.spawnItem(Ladder)
> You: player.spawnItem(Ladder)
```

Figure 2. RunJumpCode Level 3.



Run, Jump, Code!

Collected	2
Time Taken	48:04:10
Hints Used	0
Knockbacked	1
Times Reseted	0

Figure 3. Mobile App collecting myRunJumpCode statistics

The elements of this game are tailored carefully, with the aim of eliminating the possibility of the user getting deterred, confused, or bored easily. As the game progresses, it challenges players more based on an increasing learning curriculum. Each stage is focused on teaching one programming skill. The more stages the player goes through, the more skills they learn and utilize to progress. This way, anticipation is being fed to avoid boredom among players but at the same time keeping the process simple enough to cater for teenagers and young adults who have basic computer knowledge. The game provides a Code Box in which the user learn how to write a code. By using the code the player writes in the box, they are able to manipulate the object in the game. The number of stages reached should equate to the user's interaction with the computer and his acquired programming skills.

[1]https://unity3d.com/
[2]https://www.unrealengine.com/

Finally, Figure 3 shows the player's interaction data collected by the mobile application we have implemented. Users, researchers and/or parents can view the statistics for each level (picked from a drop down menu), e.g. total scores, time taken to complete the level, number of hints received, number of times reseted etc. We will be analysing this data for the upcoming study to see how engaged players are with the game in addition to assessing the improvement in their programming knowledge.

4. CONCLUSIONS & FUTURE DIRECTIONS

We presented *RunJumpCode*, an educational game to teach the fundamental concepts of programming. We used C# for this project, as we have two C# programming courses in our undergraduate curriculum. The game enhances the player's programming knowledge by providing a fun range of challenges and puzzles to solve. We promoted the interaction of programming through a 'Code Box', allowing players to enter lines of predefined code that modifies in-game objects. This tool is essential in completing the challenges and puzzles we designed. To allow alterations of its properties, we made further manipulation of each object possible, which would give the player creative freedom to complete each level. Quizzes and journals were utilized to assess and collate their learnt material for future reference. In addition, we created a mobile application to track each player's statistics throughout the game and compare their progress with other users.

Going forward, we plan to conduct an evalution study with 20-30 undergraduate tertiary students to measure the effectives of *RunJumpCode* in teaching C#. We will have a control group who will complete a pre-test, do an introductory C# course followed by a post-test. We will also have an experimental goup of students who take the same course, sit the same pre-test and will be given the game to play in their free time. They will complete the post-test at the end of the course. We will be measuring the learning outcome, engagement with game and enjoyment via pre-test, post-test and subjective questionnnaire as well as analysing the data logged during the players' interaction with the game features. We hypothesise that those students who play the game enjoy their experience and enhace their programming knowledge significantly more than their control group. We believe our research paves the way for the systematic design and development of full-fledged eductional games dedicated to teaching fundamental conceptos of programming.

REFERENCES

Baghaei, N., Nandigam, D., Casey, J., Direito, A., & Maddison, R. (2016). D*iabetic Mario: Designing and Evaluating Mobile Games for Diabetes Education*. Games for Health Journal: Research, Dev, and Clinical Applications (Vol. 5(4)), pp.270-278.

Bellemare, M.G., Veness, J. and Bowling, M. (2012) I*nvestigating Contingency Awareness Using Atari 2600 Games*. University of Alberta, Edmonton, Canada, 1-2.

Green, C.S. and Bavelier, D. (2004) *The Cognitive Neuroscience of Video Games.* Messaris & Humphreys, 5-7.

Lahtinen, E., Ala-Mutka, K. and Jarvinen, J.M. (2005). *A Study of the Difficulties of Novice Programmers.* Tampere University of Technology Institute of Software Systems. 15-17.

Milne, I. and Rowe, G. (2002) *Difficulties in Learning and Teaching Programming—Views of Students and Tutors*. Kluwer Academic Publishers, 59-62.

Pedersen, C., Togelius, J. and Yannakakis, G.N. *Modeling Player Experience in Super Mario Bros.* IT University of Copenhagen (n.d.), 1-4.

Rossing, J.P., Miller, W.M., Cecil, A.K. and Stamper, S.E. (2012). i*Learning: The future of higher education? Student perceptions on learning with mobile tablets.* Journal of the Scholarship of Teaching and Learning, 12(2), 13-16.

Ryan, J. (2011) *Super Mario: How Nintendo Conquered America.* Penguin, 155-162. Schilling, M. (2003) *Technological Leapfrogging: Lessons from the U.S. Video Game Console Industry*. California management review, 7-10.

Williams, D. A *Brief Social History of Game Play.* Univeristy of Illinois at Urbana-Chappaign (N.D), pp. 3-5. Wall-Montgomery, M. (2015) F*acebook launches TechPrep: 'By 2020 there will be 1M programming jobs left unfulfilled*'. Retrieved from http://venturebeat.com/2015/10/20/facebook-launches-techprep-by-2020-there-will-be-1m-programming-jobs-left-unfulfilled/