DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2021

Title: Investigating Student Engagement in an Educational Programming Game Utilising First-Person and Puzzle Elements

Author: Cain Rademan

Project Abbreviation: PyPOV

Supervisor(s): Gary Stewart

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	15
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	10
System Development and Implementation	*0*	*20*	15
Results, Findings and Conclusions	*10*	*20*	10
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

Investigating Student Engagement in an Educational Programming Game Utilising First-Person and Puzzle Elements

Cain Rademan
cain.rademan@gmail.com
University of Cape Town
Cape Town, Western Cape, South Africa

Abstract

Programming can be an area of considerable difficultly for many first year Computer Science students. Students are required to engage with unfamiliar constructs such as if-statements and for-loops, and develop logical thinking skills. A common issue is the problem of student engagement. Students are not engaged with their coursework enough to grapple with the challenging concepts they are faced with. A potential solution that has received significant attention in the past two decades is that of Game-Based Learning (GBL). While it is broadly established that game-based learning can be an effective tool to increase student motivation and learning, there is a lack of research into the impact of genre or specific game elements on the effectiveness of GBL software. This paper seeks to provide such an investigation by evaluating the impact that first-person and puzzle elements have on student engagement in an educational game designed to teach introductory programming. In the paper, it is detailed the design, implementation and testing of the educational game. Though limited by a small testing sample size, the results are positive, suggesting that first-person and puzzle elements can be used to increase student engagement in programming games.

CCS Concepts: • **Applied computing → Interactive learning environments**; *Computer games*; • **Social and professional topics → CS1**.

Keywords: game-based learning, programming education, pov game, puzzle game

1 INTRODUCTION

Computer scientists are some of the most in-demand workers in the modern world. According to the US Bureau of Labour Statistics's employment projections, computing careers are expected to see significant growth in the coming decade [28]. Matching this, there has been a sustained growth in Computer Science undergraduate enrollment over the last twelve years. [2]. There is however, a skills gap between the requirements of modern jobs and the abilities of university graduates [35]. Additionally, high dropout and failure rates [4] within computing degrees mean that even if students enroll, many will not go on to graduate. It is during the first year that many of these dropouts occur, and one of the key factors contributing to this is struggle with programming. Programming is a skill that is often difficult for first year students to grasp, and frustration and disenchantment in this area pose a serious problem for student engagement [20]. Misconceptions about programming, unfamiliarity with syntax and weak mathematical abilities [30], as well as lack of practice resources [4] present further barriers to students. Cultivating engagement in the first year of study is therefore critical in producing students who are both motivated to continue in Computer Science, and have the basic foundations to do so. Students need to be motivated to take control of their own learning, and grapple with the challenging concepts they will be faced with.

One approach to address the issue of student engagement that has received significant attention in recent years is that of Game-based learning. The use of educational computer games to facilitate and support student learning has been studied in a variety of contexts to varying degrees of success. Within the realm of Computer Science research, results are mixed, though largely positive. A challenge in Game-based learning research is that digital games are typically complicated and varied systems utilising a host of different game design principles that are interconnected, making it difficult to benchmark or evaluate results [19, 42].

While it is broadly established that game-based learning can be an effective tool to increase student motivation and learning, the impact of genre or specific game characteristics on the effectiveness of GBL software is not well understood, as noted by Eagle and Barnes [7]. In many papers regarding game-based learning, researchers tend to under focus on the design choices utilised in their software, and usually do not motivate the game genre, setting, or gameplay mechanics. So while in many cases researchers will conclude that a game was effective, there is a lack of attention given to what it was about their game that was compelling to learners. This poses a problem for educators wishing to leverage GBL in their teaching - it is difficult to ascertain based on the research what kind of game is best suited for their unique situation. Focused research into specific elements of games in specific domains is thus called for. As such, this paper seeks to experimentally evaluate different game design elements in isolation to determine their impact on student engagement within the field of Computer Science.

First-person games are highly popular, with millions of players worldwide. [6]. Despite the prevalence of first-person perspective, there have been few programming games developed that utilise this point-of-view, either in academic studies or in a commercial context. First-person perspective and its impact on player engagement is not well studied, though there is a general perception that utilising a first-person perspective increases immersion and interest in the game-world. There remains research to be done into whether a first-person programming game is effective in facilitating student engagement.

It is possible that part of the reason why first-person serious games are under-represented in Computer Science Education is that there seems to be little conceptual link between first-person gaming and the activity of writing code. In this project, it is hypothesised that this disconnect can be solved by creating a first-person game utilising puzzle elements to more closely match the experience of coding. In the puzzle game genre, the player is required to solve puzzles and utilise their problem solving skills. This resembles the skills required to be an effective programmer - problem solving and programming are two closely related and mutually beneficial areas [14, 32]. Since puzzle elements offer players a chance to hone their problem solving abilities, and offer logical and conceptual challenges to a player, it was hypothesised that they could be used to create a compelling first-person programming game.

This project catalogues the design, implementations and testing of an educational programming game that utilises a first-person perspective and puzzle elements. The game allows students to practice basic coding concepts, and reinforce their knowledge. The gameplay is tightly integrated with coding, where the user enters code to manipulate the game world and solve puzzles.

1.0.1 Research Question. How does the utilisation of first-person and puzzle elements affect engagement of students in an educational game designed to teach introductory programming?

2 BACKGROUND

2.1 Game-Based Learning in Computer Science Education

The field of Game Based Learning often finds applications within the realm of Computer Science. In their article "Teaching Introductory Programming from A to Z: Twenty-Six Tips from the Trenches" [43], Zhang et al. emphasise the importance of exposing first time programmers to coding in an exciting, visual way, to motivate and engage students. Games are known to be engaging [38], so can be an effective medium for accomplishing this task.

There have been several studies, with different nuances and perspectives, investigating game based learning in a Computer Science context [9, 22, 23, 40, 41, 44]. A common theme in the studies is that using games to teach Computer Science increases motivation, but the results on whether these games have a significant learning effect are mixed [22].

2.1.1 Educational Content of Existing Programming Games. In a 2018 paper "A Review of Serious Games for Programming" Miljanovic and Bradbury reviewed a number of programming games and assessed the educational content and methods by which the games were evaluated [22]. It was found that the games focused largely on problem solving and fundamental programming concepts, while there was a lack of representation of games focusing on data structures, development methods and software design. The paper mainly analysed games developed in a research context, omitting many publicly available games developed in a popular context.

Blanco and Engstrom [1] attempted to fill this gap by providing such a review of popular commercial programming games under the assumption that since these games are popular with a wide audience they have managed to provide an engaging player experience. As in Miljanovic and Bradbury's paper, they compared the games to the ACM/IEEE Computer Science curricula [34] to determine what aspects of programming they taught. Their results resembled those of Miljanovic and Bradbury, in that there was a strong representation of games teaching fundamental programming concepts, and an under representation of games teaching algorithms, design and data structures.

2.1.2 Integration between Coding and Gameplay. In programming games, an important consideration is that of the level of integration between coding and gameplay. Some programming games have tight integration, with coding systems/puzzles comprising the core mechanics of the game. An example of this type of game in the academic world would be RoboBUG [21], in which the gameplay consists of the player identifying bugs in code. Commercial examples include popular games like Code Combat (playable in browser), 7 Billion humans and Exapunks (which can be be found on Steam [37]). Others take the approach of segmenting the coding and gameplay, where the gameplay resembles that of a regular video game, but players must answer coding questions to progress between levels and continue playing. López-Fernández et al. present a game of this kind in which teachers set the questions, with the results showing increased motivation among students but not increased knowledge acquisition [18].
It can be hypothesised that games in which programming is the core mechanic are better for learning since the player

is focused on programming as the main activity. The opposite, segmented approach may not be as effective as the player may focus on the fun, non-programming part of the game while only comprehending the coding enough to pass through the level. It remains a challenge to design a programming game that is both fun, and focuses on coding as the core mechanic.

2.1.3 Educational Python Games. It has been shown that using Python to teach introductory programming concepts (instead of a language that might be considered more complex) is effective in facilitating learning [14]. We can hypothesise that because Python is simple and elegant, it is the ideal language to use in a serious game designed to teach introductory programming. There have been several studies providing implementations of games intended to do exactly this [11, 15, 33].

Escape from the Python's Den [11] is an educational game implemented within the popular game Minecraft. Unfortunately, the study from which it comes is extremely short and lacking in detail. The paper also crucially omits an evaluation of the success of the software.

Py-Rate Adventures is another such game [33]. It is a 2D platforming game which aims to teach basic programming concepts using Python. It is playable by players who have no previous knowledge in Python. In this case, the use of coding in the game is not integrated into the gameplay, but is asked in the form of questions in between levels. The player must answer the questions to progress in the game. The evaluation in the study showed that players had a positive reaction to the game, and felt that it was beneficial in terms of learning. It is important to note that the study measured perceived learning of the players and so may not be reflective of the actual learning effect of the game.

2.1.4 Point of View (POV) in Programming Games. There has been little research into point-of-view as a feature of educational games, both in the realm of Computer Science and otherwise. In a literature review conducted prior to this study [31], it was found that a large number of game based systems in research, and commercially, utilise an isometric [12], top-down [7, 10], or text-based/abstract graphics [15] approach, but there was little motivation for choice of these systems. There were no examples found that utilised a first-person perspective. Nacke et al. attempted to measure physical responses to first-person games using specialised equipment and correlated that with qualitative responses from participants [24]. However, the focus of the study was on investigating physiological responses as an indicator of psychological states of gameplay experience, and it did not compare the levels of flow produced in other perspectives. One might intuitively think that first-person games are more immersive, or yield a greater engagement then non-first-person games, but it seems that there has been little research in this area.

2.2 Puzzle Elements in Programming Games

Blanco and Engstrom provided an analysis of game genre used in programming games [1]. Their findings showed that in commercial programming games, puzzle games are the overwhelming majority. They noted that in academic papers, this is not as pronounced, although there is still a strong representation of traditional puzzle-type games. Based on the success of programming games utilising puzzle elements [7, 10, 12], there is evidence that the puzzle genre is well suited for teaching coding. This could be attributed to the fact that programming and puzzle games both involve problem solving. The link between problem solving and coding is backed up in the literature. It has been shown that programming improves cognitive reasoning skills [32]. It has also been shown that teaching problem solving before programming in an effective strategy in introductory programming courses to improve learning in students [14].

2.3 User Engagement Scale (UES)

The User Engagement Scale (UES) is a widely-used questionnaire for measuring user engagement with a system [27]. The questionnaire has overall demonstrated good reliability and validity, correlating well with other self report measures [26, 39]. The original User Engagement Scale, published in 2010, is a 31 item questionnaire measuring six factors of engagement [27], and has been used in a variety of studies/contexts. However, the findings of some studies pointed to flaws in the construction of the UES, specifically relating to its number of factors [3, 39]. In a 2018 study, O'Brien (one of the original publishers of the scale) et al. present a refined version of the UES questionnaire, along with a new short form version [25]. The refined UES reduced the six factor model of the original scale to four factors, in line with the evidence and suggestions from prior studies, and validated the new factors using statistical tools. The four factors in the refined scale are: Focused Attention, Perceived Usability, Aesthetic Appeal, and Reward. The more recent four factor model has been adopted for the purposes of measuring engagement in this study, because it addresses many of the problems that the original scale faced. A short form version of the scale, comprising twelve questions and measuring the same factors was also introduced in the paper. This was in response to one of the problems in the original questionnaire, excessive length, resulting in many studies using only select questions from the scale. This is a problem, because as O'Brien et al. note, it is difficult to assess the survey's factor structure and robustness over time and across different digital applications. The UES short form has been adopted for this study, because it will reduce the possibility of participant fatigue.

3 DESIGN

For the purposes of answering the research question, an educational programming game was developed. The gameplay consists of writing Python code to navigate a 3D game environment. The game is designed to reinforce basic programming concepts such as if-statements and for-loops, and provide a fun visual environment to practice using these constructs. The game also provides a console for new coders to see the result of their code in real time and practice debugging. It is assumed that students understand basic Python syntax and have encountered the programming constructs utilised in the game before. The intention is not to teach students coding from the beginning, but to reinforce or cement the ideas they have already learned in a visual, engaging way. The setting is that of a space rover exploring an unknown planet, avoiding obstacles and collecting valuable crystals. Seven levels were developed for the prototype.

3.0.1 Connecting first-person and coding. At the start of the design process, it was attempted to ascertain general principles for the development of the educational game, taking into account the game design elements under consideration (puzzle and first-person POV). This was challenging, as there are almost no educational programming games utilising first-person perspective and thus few examples to gain inspiration from. The lack of first-person programming games may be explained by the inherent disconnect between first-person gaming and programming. First-person games typically involve the user panning their mouse to look around an environment, and is a highly dynamic, visual activity. Coding is the opposite, a static experience in which users type text into an editor. In designing this educational game, it was attempted to solve this disconnect by utilising puzzle elements in a first-person game with the hope that a more coherent programming game experience could be obtained. Students could then be exposed to the core experience of coding within the highly engaging environment of a first-person game. However, the first-person perspective should make sense within the context of the puzzle game and not feel "tacked on" or superfluous, as resolved in the next section.

3.0.2 Exploring an environment. The idea of exploring/navigating an environment was eventually settled on as being an activity that is a good synthesis of puzzle and point of view elements. It was hypothesised that exploring an environment lends itself well to a first-person view, because there is a sense of immersion, place and spatiality. Objects of interest are revealed to players over the course of time, creating elements of suspense and spectacle. Exploring an environment also presents ample opportunities for puzzle design via obstacles, collectables and enemies. In this case the 3D space itself forms part of a puzzle environment that the player must negotiate. Taking these factors into account, the setting of a game in which the player must program their rover to navigate terrain was decided on.

3.0.3 Diegesis in games. The solution of a tech-themed space rover game also works well with the idea of diegesis. Diegesis has multiple definitions in different mediums (film, theatre, literature etc), but it typically refers to the boundary between the elements that exist within the narrative world, and the elements that are part of the telling of the story [13]. In video games, diegesis comprises the narrative game world, its characters, objects and actions. Such elements that exist in the game world are refered to as diagetic. Non-diegetic elements would be constructs like the UI or the music - they are not really part of the "real" world of the game, but are still necessary to tell the story. In this project, it is desirable that the coding be part of the diegetic world of the game, so that players do not have to suspend their disbelief when coding. For example, within a programming game with the setting of cooking, the act of coding would probably be non-diegetic, and would not make much sense. Whereas in a tech-themed space game it makes sense, coding a rover is something that can believably integrate into the game world, and it requires little suspension of disbelief.

3.0.4 Difficulty. Another consideration is the level of challenge for players. If the puzzles are too difficult, it could result in decreased self-efficacy of users, and consequently decreased engagement, as explored by Power et al. in their paper "Difficulty and self-efficacy" [29]. In another study investigating the relationship between game difficulty and several emotions [5], Chanel et al. found that high difficulty can result in higher anxiety levels in users of a game, and that if difficulty levels are too easy, the experience will become boring. Ideally, the game should be somewhere in the middle - challenging, but not too challenging. The paper also found that the engagement of a player can decrease if the game difficulty does not change, demonstrating the importance of modulating the difficulty of the game over time to match the player's increased mastery. This educational game in particular is likely to present unique challenges to new users: the notion of writing code to manipulate your character is likely to be somewhat unintuitive at first, and could illicit confusion. While it is tempting to prioritise the development of challenging and interesting puzzles, and "throw players in the deep end", it is important to make sure the experience is streamlined and teaches the user how the system works first. A solution that many puzzle games utilise is to break the game up into a number of levels of increasing difficulty (an example is the popular puzzle game Portal 2 [36]). The first couple of levels are typically simple and explain the core mechanics of the game. These principles are often modular and are reused in other levels. As the game progresses, players are expected to utilise and combine principles that they have mastered in increasingly challenging puzzles. The most effective puzzle games have just the right difficulty ramp so that as users achieve greater mastery of the system, they find that the game mirrors this in difficulty level. This is an effective solution because it reduces the chance of both boredom and confusion. This strategy of levels of increasing difficulty is the gameplay model that has been chosen for the game.

3.0.5 Gameplay Design.

Another advantage of using the above technique of modular puzzle elements is that the game designers often do not have to program large amounts of content for the game. They can simply combine the core mechanics they have introduced in interesting and surprising ways to create new levels. The challenge then is choosing core mechanics that are most likely to yield possibilities for engaging puzzles. Another desirable quality is that the game be deterministic - ie correct code will always result in winning the level and incorrect code will always result in a loss. The reason is that it is frustrating for players to input the correct code but lose due to random elements out of their control.

Taking the above considerations into account, the model for the game is as follows: Player movement is constrained to a grid that is overlaid on the terrain. This increases predictability and ensures that they cannot move in surprising ways that will break the game. Players reference neighbouring grid squares using compass directions (north, south, east, west). Grid squares can contain various entities, such as rocks, plants, and crystals. Each level has a start and end grid square. Victory is achieved by moving from the start point to the end point. Players must enter code to try and reach the end, and must write one script to do so. If their script terminates and they have not reached the end, the level will reset and they will need to attempt it again from the beginning. Players must also avoid obstacles along the way, consisting of rocks and plants. Collision with a rock or plant will result in being sent back to the beginning. They must collect all the crystals in the level to be granted victory on completion. Failure to do so will result in an unsuccessful run. Additionally, they can shoot neighbouring plants to remove them as an obstacle. They can also check what entity is in a neighbouring gridblock using a particular command.

From this, we can define the set of commands that the player will have access to:

- **Movement:** The player is able to move their rover using commands such as Rover.moveNorth()
- **Query neighbouring cells:** The player is able to check what is in neighbouring gridblocks using commands such as Rover.checkNorth("Plant")
- **Shooting:** The player is able to shoot neighbouring gridblocks, using commands such as Rover.shootNorth(). This will destroy any poisonous plants that are contained there.

3.0.6 Randomistion.

An essential skill for any programmer is the ability to write code that can handle general input that is not known before runtime. A skillful programmer can identify a pattern in a problem and write code in such a way that it solves the problem no matter what variant is presented (often through the use of conditional checks). This is reflected in the game through the inclusion of random levels. In some levels, placement of objects is semi-random (though still following some sort of pattern). The idea is that players must use "check" commands to query what is around them and decide how to handle it. In the end, only one random level was implemented, in which the player needs to check for plants around them and shoot them to move forward. However, the functionality is in place, making it an easy process to add more random levels.

3.0.7 Line limit.

Another important skill of a programmer is conciseness - writing code in its simplest and most efficient form. This often involves looking at code and figuring out what can be more efficiently done via a loop or a conditional statement. In the game, the inclusion of levels in which the player is limited in the number of lines of code they can write forces them to practice this process. In these levels, a brute force solution will not suffice as it will go over the line limit.

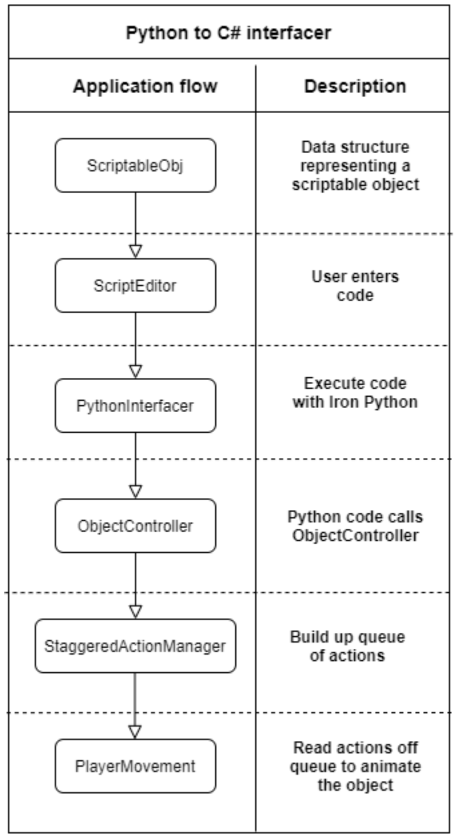
4 IMPLEMENTATION

The game was implemented using the popular real-time development platform Unity. Development followed a system of priority where the most essential and high-risk features were implemented first.

4.0.1 Python to C# Interfacer.

Since scripts in Unity are written in the language C#, and players needed to write Python code, it was necessary to facilitate the execution of Python code in a C# environment. IronPython [8] is an open-source implementation of the Python programming language which is tightly integrated with .NET (the framework upon which C# is built). Using IronPython, Python programs can integrate with applications written in other .NET programming languages, such as C#. When embedded in a C# application, IronPython supports the ability to accept a number of C# objects as part of its scope. It can then execute Python scripts at runtime that can call methods on these objects. This approach was used to implement the required functionality of users being able to call methods such as “Rover.moveNorth()” in Python, and have that reference the correct C# Unity object. The interpreter also needed to be used in a parallel study by a different researcher, Theo Thesen, focusing on narrative elements. Since this study had a different use case, the interpreter needed to be designed in a generic and modular way. The main classes in the architecture are explained below, and are depicted in a diagram (fig. 1) explaining the application flow between these classes.

Figure 1. Application flow of the Python to C# Interfacer


Python to C# interfacer
Application flow
Description
ScriptableObj
Data structure representing a scriptable object
ScriptEditor
User enters code
PythonInterfacer
Execute code with Iron Python
ObjectController
Python code calls ObjectController
StaggeredActionManager
Build up queue of actions
PlayerMovement
Read actions off queue to animate the object

ScriptableObj. ScriptableObject is a data structure storing all the information needed to make a GameObject scriptable. It contains attributes such as an ObjectController that is passed into IronPython, a ControlledObject that is the actual Unity object that is controlled by script and the spawn position and orientation it should return to when the "stop" button is hit.

ScriptEditor. The ScriptEditor is the GameObject that allows the user to input the code string. It contains a list of all the ScriptableObj's and passes these into the PythonInterfacer along with the code string when the run button is pressed.

PythonInterfacer. The PythonInterfacer contains the IronPython components necessary to interpret the user's Python code. When run is pressed, it creates a new IronPython engine, adds the ObjectController of each ScriptableObj to the scope, and sets the script source to the user's inputted string. It then executes the script, catching compile errors and outputting them to the console.

ObjectController. Instead of allowing the user to interact with the ControlledObject directly, it is necessary to have a proxy controller object. This is because the ControlledObject may have methods on it that it is undesirable for the user to access, such as, for example, a reset() method. The ObjectController is an abstract class that all controllers must inherit from, and exposes a number of methods to the user. These methods will directly call methods in the ControlledObject.

StaggeredActionManager. During the design process, it was decided that it would be more engaging for players to see their code execute over a time span, instead of in a fraction of a second. This is because the user can see the effect that each line of code has and can follow along. Since the user's Python script executes instantly, and we want to animate the Unity objects over time, it is necessary to build up a queue of actions from the script. In the current implementation, the StaggeredActionManager is the ControlledObject, and builds up a queue of actions executed by the user's script. This queue can later be read by the Unity object that needs to animate, to ensure that it performs the correct actions that the user's script defined. As it stands this component essentially runs a simple abstract version of the program beforehand, building up a queue of actions, and then uses this to animate the objects. A more ideal solution to this problem could have been wrapping the user's Python program in a thread and executing each statement line by line, as will be discussed below.

PlayerMovement. This is the Unity GameObject representing the rover. After the Python script has run, it reads the necessary actions one by one off the StaggeredActionManager. After an action has been completed, it dequeues the next action and animates based on that.

As is often the case with software engineering, upon completion it became apparent that an alternate implementation would have been preferable. As alluded to in the StaggeredActionManager section, the entire architecture could have been greatly simplified, made more stable, and more flexible, using an approach of threading. In this case, running the IronPython script in a thread, line by line, and blocking until each action is finished would have been the preferable implementation, for a variety of reasons. Crucially, in the current implementation an infinite while loop would crash the entire game, whereas in a threaded approach a player would be able to stop the code. It would be possible to facilitate the use of while-loops in the current architecture, by stopping the code if a timer exceeds a certain threshhold, say 0.5 seconds, but this is an inelegant solution. Additionally, the threaded approach would allow for game objects to move around at runtime, and still be detected by the player using, for example "checkNorth()". It is recommended that anybody looking to develop a similar system follow the threaded approach instead.

4.0.2 Environment design. Since aesthetic appeal is an important component of the engagement cultivated in a system [25], it was necessary to design an environment that

Figure 2. The Rover Model

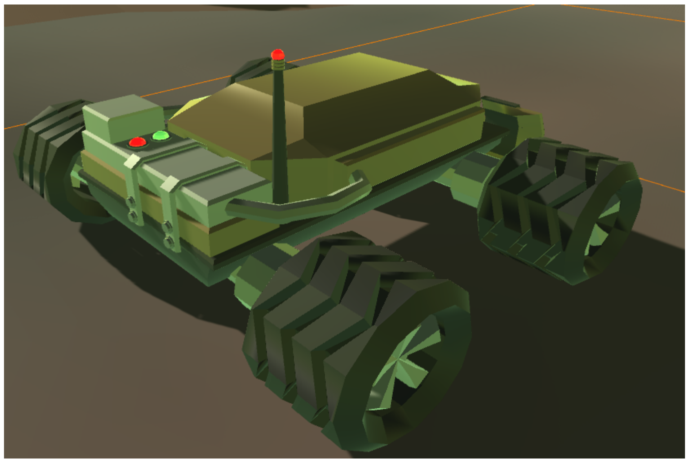
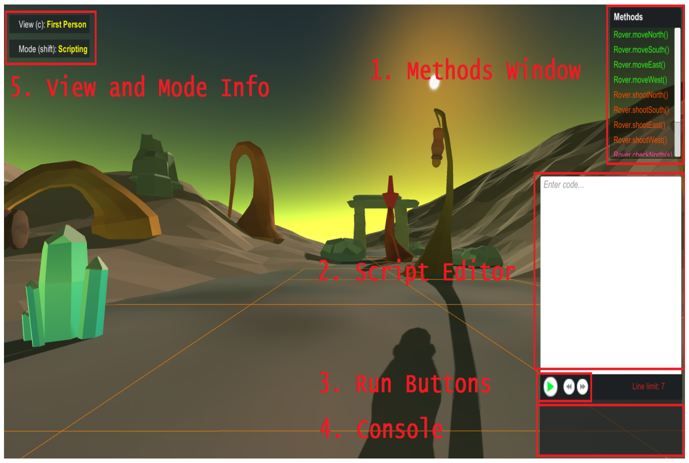


Figure 3. UI of the educational game



was attractive and interesting to look at. However, since first year students were the target group, it was also essential that the game be able to run on lower end laptops. The tradeoff between graphical quality and performance is a common problem that game developers face. Typically, improving graphics involves utilising models with a higher vertex count, or more computationally expensive shaders. Realistic graphics in particular usually come at the cost of increased computational complexity, and require higher-end computers to render at a reasonable framerate. An alternate approach to making games look more appealing is through the use of striking art direction, such as colour composition, stylistation and world building. An aesthetic that is being increasingly favoured in indie games is the "low poly" style. This style tends to forgo realism in favour of stylised low polygon count models. The low poly style was adopted for the educational game in this project. In low poly games, it is often the imagination of the player that fills in the details and brings the game world to life. The idea was to have sparse, rocky alien environments, scattered with plants and ancient ruins, contrasted by a colourful skybox, creating an effect of a strange yet visually appealing world. The assets for the project were obtained from the Unity asset store, mostly from the Low Poly Rocks Pack [17], which provided a number of low poly rocks, structures, and terrain. An exception is the rover model, which was designed by a 3D artist. There was also the consideration of the implementation of the grid over which the player can move as part of the puzzle design. The Terrain Grid System from the Unity asset store was used to efficiently draw such a grid over the terrain [16].

4.0.3 User Interface. Due to the fact that most beginner programmers have likely not been exposed to a coding game before, designing an intuitive and responsive UI was essential in ensuring user comprehension of the system. A labelled screenshot of the UI can be seen in figure 3, with each labelled component discussed below:

Methods Window. The methods window provides the user with a list of the available methods for the level. Each method can be hovered to obtain information about what it does and how to use it.

Script Editor. The user enters Python code in the ScriptEditor. It has a white background to differentiate it from the other UI components as an area in which you can type.

Run Buttons. The user can press the run button to execute the current code in the Script Editor. On click, the run button will change into a stop button, which can be pressed to reset the rover and environment. There is also a speed up and a slow button, which make the rover go faster or slower respectively, useful for allowing the user to control the pace of the game.

Console. The console area offers helpful compile and runtime output to the user, such as if their Python program fails to compile, if they collide with a plant, or the results of neighbouring grid cell querying using check commands.

View and Mode Info. These windows provide the user with visual feedback on their view and mode. The view can be either first-person or top-down. The mode can be either scripting or looking, and determines whether the focus is on the Script Editor or the environment.

5 USER TESTING

For the purposes of evaluating the system developed, user testing was conducting. Participants recruited were students whose only experience with coding was through taking an introductory programming course. These students had been exposed to Python but were still at a low programming ability level. These were mainly first year Computer Science students at the University of Cape Town, but also included a few students taking different degrees who had taken an introductory programming course. Six students in total took part in the testing. It was decided to conduct testing online in light of the risks of in-person meetings due to the COVID-19 pandemic. Students were recruited through in-person lecture visits, online announcements/emails and messages sent to Whatsapp groups.

5.1 Procedure

An hour before the experiment, participants were sent an executable version of the game specific to their computer operating system. They were informed to not begin playing the game until the experiment had begun. Participants then entered an online voice call with the researcher. First, participants played the educational game for 20-30 minutes, sharing their screen while doing so. The researcher was present while participants played to help them if they got stuck or if any bugs occurred. The researcher monitored/interacted with participants as little as possible during this time, both to test the usability of the system and to ensure that participants did not feel pressured. In some cases, if participants were struggling with the puzzles, hints were given to help them complete as many levels as possible. Afterward, the researcher and participant had a brief semi-structured conversation about their experience playing the game. Participants were prompted with questions from a set list to guide the conversation. This audio was recorded in order to obtain transcripts of what was said. Participants then undertook a survey administered through Google forms, discussed in the next section.

5.2 Survey

The survey consisted of the User Engagement Scale short form along with an additional section consisting of six questions. The two sections were separated, with the UES first. O'Brien et al.s' paper on the UES short form is useful in providing a guide to administration and scoring of the scale [25]. As suggested the wording of the questions was adapted to the context, and the order of the questions was randomised per participant (see table 1). PU-S.1, PU-S.2, and PU-S.3 were also reverse-coded as per the survey guidelines. The additional six questions (table 2) included in the survey attempted to measure factors key to the research question, such as participants' reaction to first-person and puzzle elements. Also measured were their feelings on whether the game helped to practice basic coding concepts, and would be valuable to first year Computer Science students. The additional six questions section were measured (as in the UES) with a five point Likert scale. FP-2 and PZ-2 were reverse-coded.

Table 1. The User Engagement Scale short form

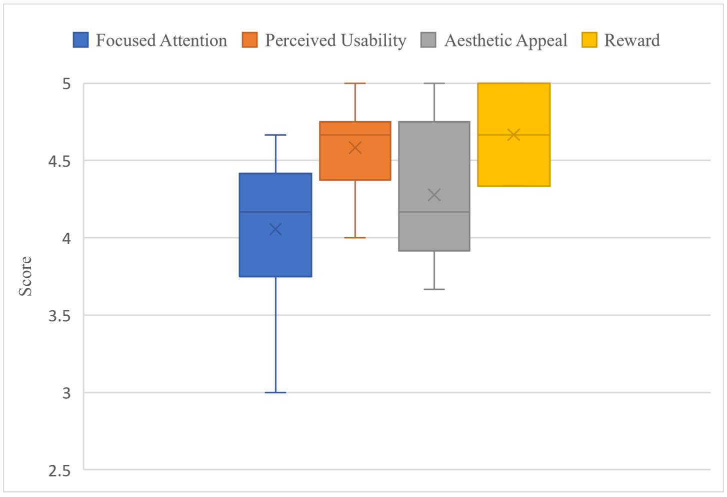
FA-S.1	I lost myself in this experience.
FA-S.2	The time I spent playing the game just slipped away .
FA-S.3	I was absorbed in this experience.
PU-S.1	I felt frustrated while playing the game.
PU-S.2	I found this educational game confusing to use.
PU-S.3	Using this educational game was taxing.
AE-S.1	This educational game was attractive.
AE-S.2	This educational game was aesthetically pleasing.
AE-S.3	This educational game appealed to my senses.
RW-S.1	Using the game was worthwhile.
RW-S.2	My experience was rewarding.
RW-S.3	I felt interested in this experience.

Table 2. Additional Questions

FP-1	The first-person view in the game increased my feelings of engagement.
FP-2	The first-person view in the game was unnecessary.
PZ-1	The puzzle solving element in the game increased my feelings of engagement.
PZ-2	The puzzles in the game were too difficult.
Q5	This experience would be valuable to first year Computer Science students
Q6	This game helped me practice basic coding concepts.

6 RESULTS

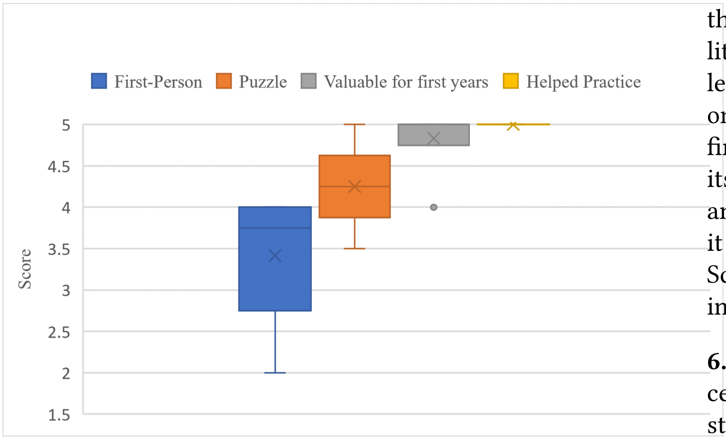
Figure 4. Scores for the 4 UES factors



6.1 User Engagement Results

The overall UES engagement score was found to be 4.39. This is a positive result and shows that overall people found the game to be engaging. The subscales Focused Attention, Perceived Usability, Aesthetic Appeal, and Reward had mean scores of 4.05, 4.58, 4.28, and 4.67 respectively. Subscale scores

Figure 5. Scores for the Additional Questions



for participants who had left out data was handled by omitting that question from the subscale mean calculation for that participant.

6.1.1 Focused Attention. Focused Attention scores were high with the exception of an outlier who felt only neutral. Generally, participants felt absorbed in the game.

6.1.2 Perceived Usability. Perceived Usability was unanimously scored at 4 or above - participants found the software to be very intuitive and easy to use. One participant felt that the interface could be improved by providing explanations of the coding constructs in the game such as if statements and for loops on the side or through a popup, so that the game could be played by coders with no experience at all.

6.1.3 Aesthetic Appeal. Aesthetic Appeal scores were also generally high. One participant noted, "I really liked the graphics style that you chose. It was kind of cute but also aesthetically pleasing. I liked the colour scheme." Another student commented on the top down camera placement, noting that it was off-centre in some levels.

6.1.4 Reward. Reward was the factor that received the highest mean score in the UES, indicating that people felt that the game produced a valuable experiential outcome. Participants unanimously indicated a five for their interest in the experience. When asked whether the game was what they had expected, many participants expressed that it was more fun than they had expected, and that they had enjoyed it.

6.2 Additional Questions Results

6.2.1 First-Person. Participants' reactions to the first-person elements were slightly positive, with a mean rating of 3.41. Overall participants were either neutral towards the first-person elements, or seemed to think that they aided engagement and were necessary to the experience. There was an outlier who indicated an FP score of 2.

6.2.2 Puzzle. Puzzle elements received a mean rating of 4.25, indicating a positive response. All students completed the first 4 levels with little difficulty, but some struggled a little on the later levels. When asked whether the difficulty level was too easy or too hard, the majority thought it was on the easier side, and would be best suited to students in the first 2-3 weeks of their course. One student noted "I think its a really good level if you're first learning how to code and getting the logic behind iteration. I wouldn't say that it would be a good level towards the end of a Computer Science course". One student thought the levels accelerated in difficulty a little too quickly.

6.2.3 Q5 (Valuable for First Years). This question received a mean score of 4.83, indicating that participants strongly felt that the experience would be valuable to first year Computer Science students.

6.2.4 Q6 (Helped Practice Basic Coding Concepts). This question received a unanimous response of 5, indicating that students felt the game was extremely effective for practicing basic coding concepts.

6.3 Improvements

Through the conversations that took place after testing, feedback was obtained about what participants felt could be improved/changed about the game. One student felt that the "limiting lines of code" for some levels could induce frustration in some students, because "people's brains think differently". They suggested not making the line limit a necessity, and instead implementing some kind of system where you are rewarded for completing the level in under the line limit. Another student noticed that, on the last level (which incorporated randomness), you could keep hitting run until you got a favourable configuration, and so make it through the level without inputting the correct code. They suggested implementing a system where, for each level, the code is run many times and has to pass multiple cases to ensure that the player did not "get lucky". Another student commented on the fact that while-loops were unusable, suggesting that it would be a good feature to have. During one student's run, a strange bug occured in which plants shot by the player would not be properly destroyed, and the player's code would halt at this point. This bug could not be reproduced, and is of high priority to fix.

7 DISCUSSION

As the sample size in the study was small, sweeping conclusions cannot be drawn from the data. More testing would need to be done to reproduce and confirm the results obtained. It is reasonable however, to do some tentative analysis of the current results, bearing in mind that they may be inaccurate. The original research question involved ascertaining the effect of first-person and puzzle elements on

engagement. Based on the results of the survey, the game seems to be engaging to students, as measured by the User Engagement Scale. Students also felt that the first-person and puzzle elements increased their feelings of engagement. This was a self-report however, so it is unclear whether engagement was really increased by these factors or whether some other properties of the game facilitated engagement. It may be that a similar programming game without these elements would have resulted in a more engaging experience. However, the engagement generated does seem to be to some extent aided by the first-person and puzzle elements - these elements were designed to be core components of the game, and so it makes sense that they must influence the engagement generated while playing.
In terms of the relative engagement factor of the two elements, puzzle received a higher score than first-person, indicating that puzzle was the element with a higher engagement facilitation.

The results pertaining to whether the game could facilitate the practice of basic coding concepts, and whether participants thought it would be beneficial to first year programming students are particularly encouraging, with very high scores. When asked in what way the game would be beneficial in introductory programming courses, students expressed that playing the game was a lot more enjoyable than the traditional methods of teaching. One student is quoted as saying "I remember in my first year Computer Science course I found the assignments (which is where you really consolidate how to code) could often be really stale. I think this could be a way for people to develop there skills with coding but in a much more enjoyable way."

An interesting thing to note is that many students specifically mentioned that they thought the game was a good introduction to the logic of if-statements and for loops, and would be best suited for students in their first couple of weeks of study. It was originally intended for the game to be of a slightly higher level, incorporating additional constructs such as variables and primitive data types, and while-loops. During development, the focus was shifted to providing a more fundamental experience that requires less prior knowledge and focuses on coding logic. The responses of students are encouraging and indicate that the game achieved these goals. Some students did indicate that they thought that it would be good to add more breadth of content, so that "as you go forward you could play more levels". Some students also thought that adding more challenging puzzles would be good for the students who want extension.

7.1 Limitations

The results of this study are limited by the small sample size and lack of a control group. An ideal testing scenario would be one in which different versions of the same programming game were developed: with first-person elements, with puzzle elements, with both of these and with neither. This would enable insight into the effects of first-person and puzzle on engagement in isolation, as well as their combined effect and the interplay between the two elements. Such a rigorous procedure was beyond the scope of this research. Should such a study take place, based on the results of this paper it is recommended that it be hypothesised that first-person and puzzle elements have a positive effect on engagement in educational programming games.

Another potential limitation of this study to bear in mind is that testing was done with students who had already completed an introductory programming course. As such, students were at a higher coding level than the game was designed for. As such, it is difficult to ascertain whether the the puzzles are at the right level of difficulty. Many students indicated that the difficulty would be suitable for the first couple of weeks of a course, but they may have been overestimating their own abilities in the past. Some of the students actually struggled with the later levels, even though they had already completed an introductory programming course. It would be ideal to conduct a study with participants who are actually in the midst of an introductory programming course, to more accurately judge the level of difficulty of the software.

8 CONCLUSIONS

This paper saw the development of an educational Python game designed to allow students to practice introductory programming concepts. To achieve this goal, the paper saw the successful implementation of a modular system for interpreting players' Python code at runtime in a Unity environment. This system was leveraged within Unity to enable the core code writing mechanics of the game. The mechanics and setting of the game were motivated using game design principles, and it was attempted to integrate first-person and puzzle elements into the game to create compelling coding gameplay. User testing showed positive results, suggesting that the educational programming game developed is effective in eliciting feelings of engagement in students. Students responded positively to the first-person and puzzle elements, and linked them to their feelings of engagement. They felt that the first-person elements were necessary to the game, and that the difficulty of the puzzles was suitable for new Computer Science students. They also felt that the game helped them practice basic coding concepts, and would be valuable to first year students. The findings of this study should in no way be treated as definitive, and it is recommended that more detailed study take place to reproduce and confirm the results. It is hoped that this paper will bolster similar research, and encourage more comparative study of different game design elements in educational programming games.

References

[1] Ander Areizaga Blanco and Henrik Engström. 2020. Patterns in Mainstream Programming Games. *International Journal of Serious Games* 7, 1 (Mar. 2020), 97 – 126. https://doi.org/10.17083/ijsg.v7i1.335

[2] Computing Research Association. 2019. 2019 Taulbee Survey.

[3] F Banhawi and N. M Ali. 2011. Measuring user engagement attributes in social networking application. In *2011 International Conference on Semantic Technology and Information Retrieval*. IEEE, 297–301.

[4] Theresa Beaubouef and John Mason. 2005. Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations. *SIGCSE Bull.* 37, 2 (June 2005), 103–106. https://doi.org/10.1145/1083431.1083474

[5] Guillaume Chanel, Cyril Rebetez, Mireille Bétrancourt, and Thierry Pun. 2008. Boredom, engagement and anxiety as indicators for adaptation to difficulty in games. In *Proceedings of the 12th international conference on entertainment and media in the ubiquitous era (MindTrek '08)*. ACM, 13–17.

[6] Valve Corporation. [n.d.]. *Steam & Game Stats*. Retrieved June 22, 2021 from https://store.steampowered.com/stats/

[7] Michael Eagle and Tiffany Barnes. 2009. Experimental evaluation of an educational game for improved learning in introductory computing. In *Proceedings of the 40th ACM technical symposium on computer science education (SIGCSE '09)*. ACM, 321–325.

[8] .NET Foundation. [n.d.]. *IronPython*. Retrieved June 22, 2021 from https://ironpython.net/

[9] Stefanos Galgouranas and Stelios Xinogalos. 2018. jAVANT-GARDE: A Cross-Platform Serious Game for an Introduction to Programming With Java. *Simulation gaming* 49, 6 (2018), 751–767.

[10] Andrew Hicks. 2010. Towards social gaming methods for improving game-based computer science education. In *Proceedings of the Fifth International Conference on the foundations of digital games (FDG '10)*. ACM, 259–261.

[11] Ilish Kane, Chuy-Thuy Pham, Adam Lewis, and Vanessa Miller. 2019. Escape from the Python's Den: An Educational Game for Teaching Programming to Younger Students. In *Proceedings of the 2019 ACM Southeast Conference (ACM SE '19)*. ACM, 279–280.

[12] Cagin Kazimoglu. 2020. Enhancing Confidence in Using Computational Thinking Skills via Playing a Serious Game: A Case Study to Increase Motivation in Learning Computer Programming. *IEEE Access* 8 (2020), 221831–221851. https://doi.org/10.1109/ACCESS.2020.3043278

[13] Erica Kleinman, Elin Carstensdottir, and Magy Seif El-Nasr. 2019. A Model for Analyzing Diegesis in Digital Narrative Games. In *Interactive Storytelling*. Springer International Publishing, Cham, 8–21.

[14] Theodora Koulouri, Stanislao Lauria, and Robert D Macredie. 2015. Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM transactions on computing education* 14, 4 (2015), 1–28.

[15] Dimitra Koupritzioti and Stelios Xinogalos. 2020. PyDiophantus maze game: Play it to learn mathematics or implement it to learn game programming in Python. *Education and information technologies* 25, 4 (2020), 2747–2764.

[16] Kronnect. [n.d.]. *Terrain Grid System*. Retrieved Aug 2, 2021 from https://assetstore.unity.com/packages/tools/terrain/terrain-grid-system-47215

[17] LMHPOLY. [n.d.]. *Low Poly Rocks Pack*. Retrieved Aug 2, 2021 from https://assetstore.unity.com/packages/3d/environments/low-poly-rocks-pack-70164

[18] Daniel López-Fernández, Aldo Gordillo, Pedro P. Alarcón, and Edmundo Tovar. 2021. Comparing Traditional Teaching and Game-Based Learning Using Teacher-Authored Games on Computer Science Education. *IEEE Transactions on Education* (2021), 1–7. https://doi.org/10.1109/TE.2021.3057849

[19] Katie Larsen McClarty, Aline Orr, Peter M Frey, Robert P Dolan, Victoria Vassileva, and Aaron McVay. 2012. A literature review of gaming in education. *Gaming in education* (2012), 1–35.

[20] A McGettrick. 2005. Grand Challenges in Computing: Education–A Summary. *Computer journal* 48, 1 (2005), 42–48.

[21] Michael Miljanovic and Jeremy Bradbury. 2017. RoboBUG: A Serious Game for Learning Debugging Techniques. 93–100. https://doi.org/10.1145/3105726.3106173

[22] Michael Miljanovic and Jeremy Bradbury. 2018. A Review of Serious Games for Programming.

[23] Mathieu Muratet, Patrice Torguet, Jean-Pierre Jessel, and Fabienne Viallet. 2009. Towards a serious game to help students learn computer programming. *International journal of computer games technology* 2009, 1 (2009), 1–12.

[24] Lennart E Nacke and Craig A Lindley. 2010. Affective Ludology, Flow and Immersion in a First- Person Shooter: Measurement of Player Experience. (2010).

[25] Heather O'Brien, Paul Antony Cairns, and Mark Hall. 2018. A practical approach to measuring user engagement with the refined user engagement scale (UES) and new UES short form. (2018).

[26] Heather L O'Brien and Mahria Lebow. 2013. Mixed-methods approach to measuring user experience in online news interactions. *Journal of the American Society for Information Science and Technology* 64, 8 (2013), 1543–1556.

[27] Heather L O'Brien and Elaine G Toms. 2010. The development and evaluation of a survey to measure user engagement. *Journal of the American Society for Information Science and Technology* 51, 1 (2010), 50–69.

[28] US Bureau of Labour Statistics. 2019. Employment Projections 2019-2020. https://www.bls.gov/emp/tables/emp-by-detailed-occupation.htm

[29] Jason Power, Raymond Lynch, and Oliver McGarr. 2020. Difficulty and self-efficacy: An exploratory study. *British journal of educational technology* 51, 1 (2020), 281–296.

[30] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017), 24 pages. https://doi.org/10.1145/3077618

[31] Cain Rademan. 2021. Literature Review of Game-Based Learning in Computer Science Education. (2021), 3–4.

[32] Ronny Scherer, Fazilat Siddiq, and Bárbara Sánchez Viveros. 2019. The Cognitive Benefits of Learning Computer Programming: A Meta-Analysis of Transfer Effects. *Journal of educational psychology* 111, 5 (2019), 764–792.

[33] Grigorios Sideris and Stelios Xinogalos. 2019. PY-RATE ADVENTURES: A 2D Platform Serious Game for Learning the Basic Concepts of Programming With Python. *Simulation gaming* 50, 6 (2019), 754–770.

[34] The Association for Information Systems (AIS) The Joint Task Force: Association for Computing Machinery (ACM) and IEEE Computer Society. 2020. Computing Curricula 2005: The Overview Report.

[35] The Association for Information Systems (AIS) The Joint Task Force: Association for Computing Machinery (ACM) and IEEE Computer Society. 2020. Computing Curricula 2020.

[36] Valve. [n.d.]. *Portal 2*. Retrieved Sept 14, 2021 from https://store.steampowered.com/app/620/Portal_2/

[37] Valve. [n.d.]. *Steam*. Retrieved June 3, 2021 from https://store.steampowered.com/

[38] Eric N Wiebe, Allison Lamb, Megan Hardy, and David Sharek. 2014. Measuring engagement in video game-based environments: Investigation of the User Engagement Scale. *Computers in human behavior* 32 (2014), 123–132.

[39] Eric N Wiebe, Allison Lamb, Megan Hardy, and David Sharek. 2014. Measuring engagement in video game-based environments: Investigation of the User Engagement Scale. *Computers in human behavior* 32 (2014), 123–132.

[40] Stelios Xinogalos. 2018. Programming Serious Games as a Master Course: Feasible or Not? *Simulation gaming* 49, 1 (2018), 8–26.

[41] Mirac Yallihep and Birgul Kutlu. 2020. Mobile serious games: Effects on students' understanding of programming concepts and attitudes towards information technology. *Education and information technologies* 25, 2 (2020), 1237–1254.

[42] Michael F Young, Stephen Slota, Andrew B Cutter, Gerard Jalette, Greg Mullin, Benedict Lai, Zeus Simeoni, Matthew Tran, and Mariya Yukhymenko. 2012. Our princess is in another castle: A review of trends in serious gaming for education. *Review of educational research* 82, 1 (2012), 61–89.

[43] Xihui Zhang, John D Crabtree, Mark G Terwilliger, and Janet T Jenkins. 2020. Teaching Tip: Teaching Introductory Programming from A to Z: Twenty-Six Tips from the Trenches. *Journal of information systems education* 31, 2 (2020), 106–.

[44] Dan Zhao, Cristina Hava Muntean, Adriana E Chis, and Gabriel-Miro Muntean. 2021. Learner Attitude, Educational Background, and Gender Influence on Knowledge Gain in a Serious Games-Enhanced Programming Course. *IEEE transactions on education* (2021), 1–9.