ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

HCI Master

Masters in Human-Computer Interaction – University of Patras

Joint Postgraduate Program of Electrical and Computer Engineering (ECE) and Computer Engineering and Informatics Department (CEID)

Postgraduate Thesis

"Effectiveness of serious game designs for software developers"

by

Kallirroi Zarkadoula

Major: Human Computer Interaction

Supervising Committee:

Xenos Michalis, Professor (supervisor)

Komninos Andreas, Assistant Professor

Mavrommati Irene, Associate Professor

Patra, 2021

Abstract

Games are generally considered to be motivating and engaging. Nowadays, people, especially the young, spend a lot of time playing different subgenres of games. Serious games, games that expand the traditional notion of gaming and have a different purpose other than pure entertainment, can be used in Computer Science education and propose a different type of learning and discussing relevant topics. Games are not only used in many ways in computer science education but also consist of a great part of students' extracurricular activities. The main purpose of serious games sessions is to put their suitability to the test and discuss whether they can be considered as an efficient teaching method.

The challenge of digital serious games is to identify which of these are most effective in meeting educational outcomes, enriching computing education and fitting into existing teaching methods. In serious games, we can define points of interest like player's role, functionality, level of engagement, connection to other players, etc. Using and tweaking these dimensions we can thereafter produce not only an educational outcome but also an approach towards Human-Computer Interaction (HCI) methods and techniques.

This Master Thesis targets to analyze serious games for Computer Science education in two ways: 1) for their design elements and 2) for the evaluation of their effectiveness. While thinking of the design elements, one can decide upon them by analyzing the mechanics, dynamics, and aesthetics of the game. All of them are essential and valuable parts of game design and comprise interconnected and interrelated concepts. Also, the main aim of the study is to focus on the design, development, and evaluation of different subgenres of games.

The initial approach is to design, develop and evaluate games that will consist of different interfaces, but both are designed and developed for software developers and target accordingly fundamental programming concepts. Participants of the experiments will be users with an elementary knowledge of programming that will be invited to play one of the subgenres of the game with randomly assigned interfaces and activities.

The effectiveness of each game genre will be evaluated concerning the interface by using experimental methods and conducting interviews with the participants asking their opinions regarding the comprehensibility of each game genre. All collected data will be used for revisions and improvements if needed, and the results of the evaluation will be used to summarize and conclude to what extent the defined goals are achieved.

Acknowledgements

Throughout this study, I received a lot of support and assistance. Firstly, I would like to thank my supervisor, Professor Michalis Xenos, whose assistance and expertise was undoubtedly valuable during the writing of this thesis. His guidance was decisive, while his suggestions and useful comments provided great feedback in the process of the research. Also, I am deeply indebted to the participants who took part in the experiments of this study. Finally, my sincere gratitude to my family and my friends for their help and support during this journey.

Table of Contents

1 Introduction 1
1. 1 Objective 1
1. 2 Structure 1
2 Serious Games and Software Development 3
2. 1 The Principles of Computer Games 3
2.2 Game Elements 4
2.3 Computer Games for Software Development 4
2.4 Problems with Learning Programming 5
2.5 Case Studies 5
2.6 Review of Case studies 25
2.7 Learning Outcome 25
3 Game Design 27
3.1 Conceptualization 27
3.2 Game Design 28
3.3 Research Questions 30
3.4 Decision of game engine 31
3.5 Unity and C# 32
3.5.1 C# 32
3.5.2 Unity 33
4 Game Implementation 35
4.1 Game's overview 35
4.2 Environment / Player / Other components 35
4.2.1 Game Environment 35
4.2.2 Player / Avatar 37
4.2.3 Other components 38
4.3 Game Mechanics 39
4.3.1 Levels 39
4.3.2 Timer 41
4.3.3 Score 41
4.3.4 Help 42
4.3.5 Hint 44
4.3.6 Clear 45
4.4 Versions 45
4.4.1 Drag-and-drop / Production 45
4.4.2 Drag-and-drop / Debugging 47

4.4.3 Typing / Production 48
4.4.4 Typing / Debugging 50
4.5 Game's Challenges 51
4.5.1 Level 1 51
4.5.2 Level 2 52
4.5.3 Level 3 53
4.5.4 Level 4 54
4.5.5 Level 5 55
4.6 Technical Limitations 55
5 Experimental Analysis 58
5.1 Design the experiment 59
5.2 Metrics 59
5.2.1 Effectiveness 60
5.2.2 Efficiency 60
5.2.3 Usability 60
5.3 Limitations 61
5.4 Experiment protocol 61
5.5 Experimental Analysis 63
5.5.1 Demographics 63
5.5.2 Results 64
5.5.2.1 Test for Normality 64
5.5.2.2 RQ1 65
5.5.2.3 RQ2 66
5.5.2.4 RQ3 68
5.5.2.5 RQ4 68
5.5.3 Interview Findings 69
6 Discussion and Future Work 71
6.1 Main Conclusions 71
6.2 Limitations 72
6.3 Future Work 73
References 74

1 Introduction

Human-computer interaction (HCI) surfaces in the 1980s with the advent of personal computing when computers were no longer room-sized, expensive, and built for experts but started being accessible to more and more people. Consequently, the need to create human-computer interaction as easy and efficient as possible for less experienced users became increasingly vital. Also, with the advent of personal computing, fields like computer science and software development started rising and nowadays are more popular than ever before. To keep up with new technologies in software development, the need for further educational methods and approaches was created.

HCI research has provided many methods for ensuring good usability during software development, but the importance of usability in games has been acknowledged [1]. Computer games have grown above and beyond simple activities. So, they were proposed as a viable solution for software developers to improve their skills and practice notions they struggle with. At the same time, HCI started extending its methodologies to the field of Serious Games to help researchers understand better the interactions among the players and the technology and assess the metrics related to the player like satisfaction, effectiveness, learning outcome, etc.

1. 1 Objective

The dominant focus of HCI, concerning serious games, is on the player's positive and fun experience, effectiveness, efficiency, and usability of the design. However, there is a lack of research examining the usability of activities during actual development in serious games. At the same time, the game development industry, although growing fast, seems to not have enough resources to invest in broadening the field of serious games concerning software development.

This study reviews the existing serious games and research on usability issues and examines the effectiveness of different designs. An interesting perspective while designing a serious game for software developers is to think of more creative ways that a player can interact with the system. To design meaningful games with content relevant to computer science topics, it is important to study which are the main problems in learning programming, which notions seem difficult to programmers, and especially what are the main tasks and activities a software developer performs.

The subject of this thesis not only is based on the theoretical background of existing games but also examines usable metrics based on the implementation of a serious game that follows this study. So, to have the freedom to customize, analyze and examine further details on the usability and effectiveness issues, different interfaces and tasks are proposed and implemented.

1. 2 Structure

Firstly, Chapter 2 expands on the principles and game elements of computer serious games, and how they can be linked to software development as a learning process. Also, common problems with learning programming are analyzed and case studies are described in detail to conclude in a final review on the learning outcome of the games and the game elements that lead to this outcome. Chapter 3 explains the overall game design, the process of ideation, and the decision on the technology and the programming language that is used for the

implementation. Chapter 4 will showcase the game's implementation including the mechanics, the different versions that were implemented, the tasks in the game, and the technical limitations. Chapter 5 will provide a glimpse into the design of the experiment, the limitations, and the experimental analysis and finally Chapter 6 outlines conclusions and discusses the limitations.

2 Serious Games and Software Development

Software development and computer science are more popular than ever before and the number of students or adults that are interested in diving into this field is ever more increasing. To keep up with this need, the approach of educational computer games has been present for decades and has been established. Below we introduce the principles of computer games and general game elements. These principles are further linked to programming to produce efficient computer games for software developers. Towards this approach, we also tackle the challenge of the variety of students' backgrounds in programming and discuss case studies of modern and outdated serious games that have been developed until now. A brief review of the case studies and an analysis of their learning outcome is presented.

2. 1 The Principles of Computer Games

Nowadays, programming and software development have been established as essential skills that everyone should have or acquire and have been widely proposed in education. Websites like code.org popularize coding as an efficient path to jobs and prosperity and most countries have also introduced plans to teach children and students basic principles of computational thinking and software development. Also, there is widespread agreement about the importance of computational thinking and how it should be integrated into educational programs. Computational thinking is an approach to problem-solving that draws on techniques used in computer science [2].

In formal education, we notice a shift from the traditional didactic model, which is focused on instruction, to a model that emphasizes more on the learners' needs and their active engagement in the learning process. At that point, computer games can stand efficiently as there is some empirical evidence that games can be effective tools for enhancing learning and understanding of complex subject matter [2] [3]. To build an instructional game, we must first define the characteristics and the model of instructional games. There is a tacit model that is inherent in most studies of instructional games. Firstly, the objective is to design an instructional program that incorporates certain features or characteristics of games. Secondly, we should assure that the features we defined can trigger enjoyment, interest, user behaviours like greater persistence or time on task, and further system feedback [4]. The model proposed is depicted in Figure 1. In this model, we also note that the game cycle is iterative and involves repeated judgment -behaviour-feedback loops.

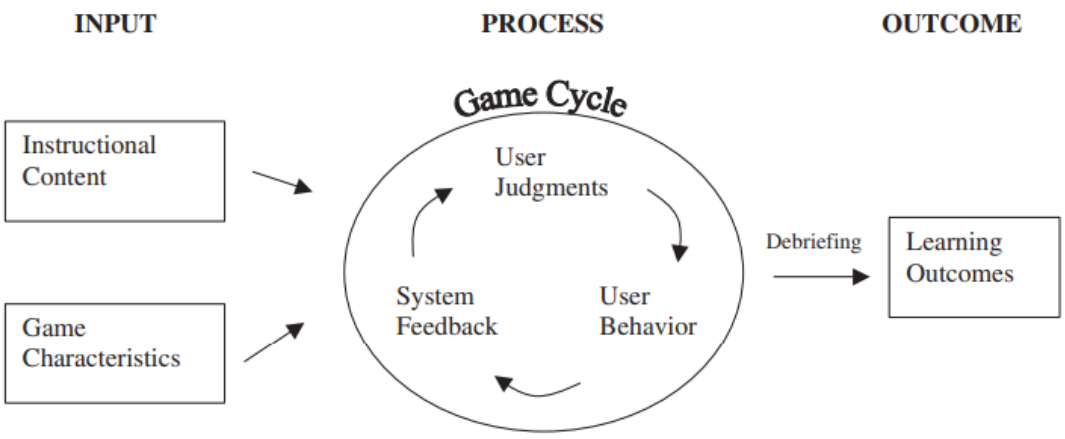

Figure 1: Input-Process-Outcome Game Model [5]

Furthermore, to refine this instructional model and the theoretical formulations of effective instruction, it is important to note the characteristics that constitute a game. The game characteristics can be described in terms of six broad dimensions: fantasy, rules/goals, sensory stimuli, challenge, mystery, and control [4]. Games can also be social environments and there is a great range of serious, or not, games that involve large distributed communities. A game overall can be characterized by a system of interaction that possesses abstract qualities like player intention, perceivable consequence, and narrative [6].

Gaming environments have tremendous appeal and can be motivating and engaging outside the laboratory settings. Good serious games provide players immersive situations where they can get completely involved in the context. Players feel motivated and improve their understanding of the world and the environment. They can also see the relevance between the real world and the virtual world. So, when they encounter notions of everyday life (like mathematics, programming, physics, etc.), not only they do not experience them anymore as cold and abstract but also have a positive experience with them by playing a game and receiving satisfaction.

2.2 Game Elements

To analyze existing games and design an effective game for software development learning, it is important to study the game elements and establish a common vocabulary. So, some key principles and notions are described below as mentioned in [5]:

- Player: The system requires a player which it interacts with. There are usually multiple operators, who can be people or computers.
- State: The system has different conditions.
- Rules: Rules are operational, constitutive, and implied methods of state transition [7]. They control the setup of the game, the progression of play, and the resolution.
- Sequence: The flow of state transition. Some systems are turn-based, while others are real-time, or some combination.
- Representation: The system has the means to represent its state using tokens.
- Goal: The end-state of the system, which the players strive towards.
- Decisions: The system must present players with “a series of interesting choices” [8]. Often this by presenting obstacles or providing some form of challenge.
- Interface: The system requires an interface that facilitates feedback (output) and response (input)

2.3 Computer Games for Software Development

While traditional instruction focuses on concepts and procedures, game-based learning in computer science can improve the ability of the learner to learn outside of the context [5]. At that point, we examine briefly the meaning and the value of game development for computer science education and software development. Games can not only be a valuable tool for enriching software development education but also provide the instructor with the ability to build a playful and motivating learning environment and collect learning metrics easily.

There is a unique relationship between digital games and computer science since programming and software development itself contains many of the same elements found in games [5]. In 1980, Thomas W. Malone [9] stated that “In some senses, computer programming itself is one of the best computer games of all” establishing a direct relationship between computer games and software development and computer science generally.

Educational technology can provide structured learning environments in the absence of direct mentorship [10]. Approaches vary from puzzle games, tutorials, and cognitive tutors [11]. The approach of puzzle games will be further developed and discussed through the analysis of various puzzle games. Tutorials are broadly introduced on the website of Codecademy[1] where the student can select from a wide range of the software development skills he wants to acquire.

2.4 Problems with Learning Programming

Towards this approach, various alternatives have been introduced and many learning environments that teach computing science issues and programming have been presented. Though, learning to program in universities is significantly challenging as it aims at students with varying academic backgrounds. There are two main categories of serious games around software development: games that target mainly software developers and games that are characterized more by a psychological/educational approach. The second category focuses on comprehension of the basic concepts, mental models, and identifying critical notions for novice programmers, etc.

To tackle this issue, we find the notions that are difficult for students to understand. Several types of research try to identify the concepts that novice programmers and developers struggle to become familiar with. In the field of psychology of programming, the research findings show that students who learn programming have a problem understanding complex and cognitively demanding concepts. Some competencies are composing the programming knowledge domain and a designer of a serious game should be able to adequately address them. Efficient serious games must focus on the problems the novice programmers face and provide them with a pleasant and educational experience. Novice programmers have problems with not only loops and conditionals but also recursive functions [12]. At the beginning of the learning process, there is an emphasis on the syntax of the language, and thereafter the focus shifts to semantics [13]. Though, we note that novice programmers struggle in learning the syntax of a certain language. So, it would be interesting to make syntactic errors impossible in a game so that the student would not waste time finding the error that is caused by the syntax.

2.5 Case Studies

In [14] the authors introduce a serious game based on the presentation of variables in programming. The game relies on the visualization of different types of variables and the interpretation of the assignment sentence. It also encourages interactivity and deeper learning and understanding of the role and the importance of variables. The authors also propose a new method for teaching programming by presenting the main ideas behind the programming concepts without overloading the player with the syntax of a certain programming language. In that way, the student can focus on the meaning of the concept and not on the way that a certain command should be written and compiled. As the game focuses on the value of the variable in software programming its main objective is that the student understands the variable as a value that is stored in the memory of the computer and learns that different types of variables are not compatible. The student should also learn that variables have their labels and learn to predict their values after several assignments.

[1] https://www.codecademy.com/

The authors implemented a prototype of the game using Flash with Actionscript 3 to make the game easily accessible to the audience. The game scenario takes place on a planet where chaos has dominated, and the inhabitants need to transport some goods to other planets. The goods that they transport have similar properties as variables in software development. In that way, a metaphor is used so that students can learn the concept of the variable in an intuitive environment where there are no strict constraints about the commands and the way that they should be written.

The game consists of four parts and all of them describe a situation linked to the state of the planet. While playing the first and second game, the student should learn how to declare a variable and understand that there are several types of variables (Figure 2). During the third game, the player should now feel convenient with the initialization of a variable and its storage in a memory location. Finally, the student executes the assignment of an implicit value to a variable (Figure 3).

Figure 2: First and second game

Figure 3: Third and fourth game

The Foos is a commercial game that was developed by codeSpark[2] and addresses the basic programming concepts. The target players are students 5-10 years old and the game does not require any reading skills since it aims at young children. The player has the form of an avatar in the game which performs a series of moves to win the final prize. In Figure 4, the player jumps above the wooden boxes to win the doughnut that represents the prize. There are also some other collectable items which are bonus rewards, and the player can pick them up along the way to the final prize.

[2] http://codespark.org

Figure 4: The Foos

Lightbot is an educational game in Flash developed by Armor Games and endorsed by Code.org[3] as an Hour of Code exercise. It is available as an online game for iOS and Android platforms. It also supports two versions, one for students aged 4-8 and one for students aged 8 and above. In this game, the environment is presented with tiles and the player has the avatar of a robot as depicted in Figure 5. The player's goal is to direct the robot to traverse the grid by producing a sequence of moves. To produce the final sequence, he has some available instructions like move, turn, light, and jump. Since the player has formed the sequence, he runs the program to step through the programmed sequence. The objective of the game is to have the robot reach the dark blue squares in the grid and light them up. The solution for each level is an algorithm that solves that level and produces the expected sequence.

As most levels in the game are based on repetition and repetitive sections, this functionality can be leveraged by placing repeated actions in functions. The designers have introduced an interesting constraint in the game. The space for the tile sequences is limited, forcing the player to consider wisely the code capacity. The model of LightBot, a visual representation of programming, makes it attractive [15]. Students must understand how the model works and interact with it. Also, notation could be introduced in the game so that the student could see a rough representation of the solution of each level.

In [15] the authors perform an evaluation of Light-Bot where the overall score of the game is satisfying with strengths in particular areas. The authors found Light-Bot to be a useful game for practising computational thinking and was shown to be stronger in particular conceptual areas like models and abstractions and tools and resources. Though, it has a weak performance in areas like patterns and algorithms and does not provide flexible tools.

[3] https://code.org/

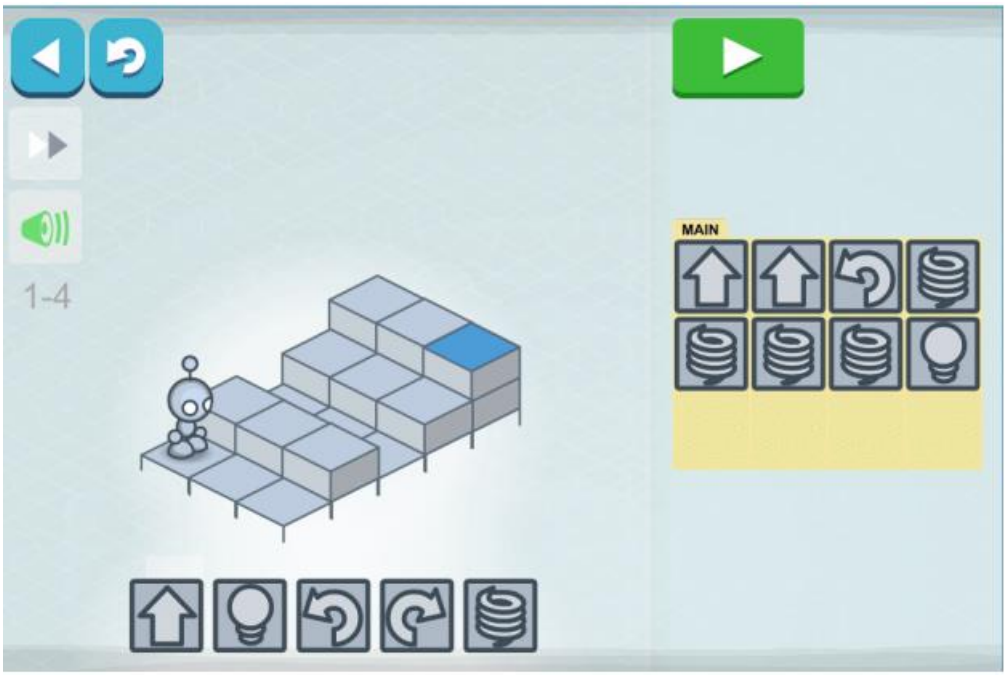

Figure 5: Lightbot

Human Resource Machine is a commercial game published by Tomorrow Corporation in 2015 and it is available for Windows, Mac, Linux, and Android platforms. The game opens outside the drab-coloured office building, whereupon the player selects an employee avatar. In the first level of the game (Figure 6) the player is asked to compose a sequence in the program editor panel on the right that outputs the expected results. A manager is sitting at a desk and has assigned to the avatar a task. The player must construct a sequence of moves that the employee should follow to accomplish the objective. The objective of the mission is to move all the boxes from the 'In' conveyor belt to the 'Out' one. The avatar executes an assembly program that outputs the maximum of each input pair.

If the player does not move successfully all the boxes from the 'In' to 'Out' conveyor belt by producing a sequence that outputs the expected tiles, the manager reprimands the employee.

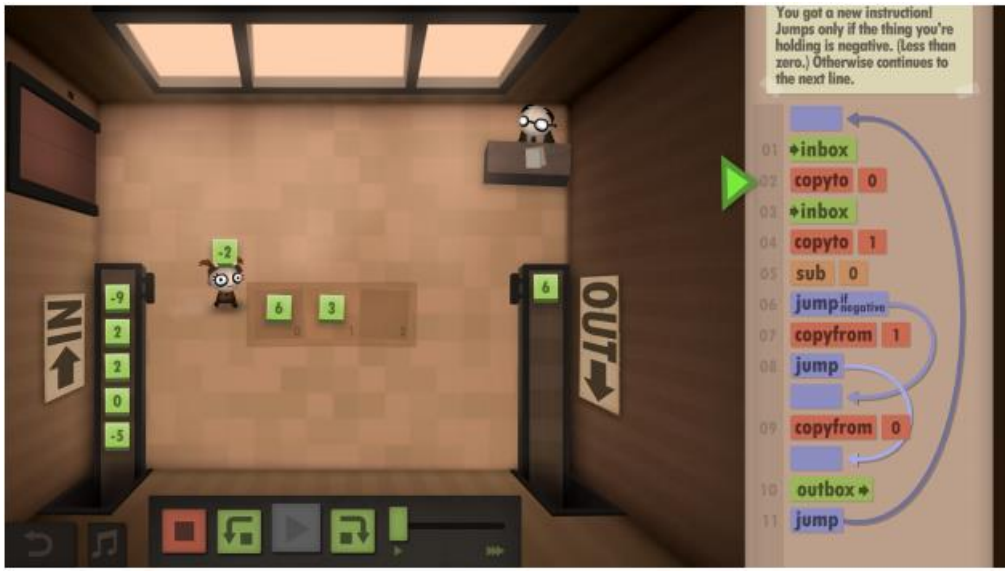

Figure 6: Human Resource Machine

In [16] the authors designed a general-purpose concurrent programming system, ToonTalk, in which the source code is animated, and the programming environment is a video game. ToonTalk follows design principles that dictate the encouragement of exploration and curiosity of the players and has borrowed some design elements from video games. The programmer controls a "programmer persona" in the video game to construct, run, debug and edit programs. The authors also support that the game not only is a self-teaching system but also provides a rich, flexible and expressive environment. Also, it is very easy and pleasant to learn, and at the same time, it can be a powerful tool for programming. Those attributes are hard to combine as the more flexibility a programming language offers the more effort someone has to put in to learn its capabilities and limitations and become familiar with them.

The authors approach this constraint in a very interesting way. The game uses a consistent set of concretizations to describe concepts that exist in concurrent systems. The authors also support their decision on a concurrent and not a sequential system as the model of the world is concurrent and it would be easier for the children to be exposed to naturally driven rules and systems.

The user in the game can explore the city's environment with robot helpers. There are helicopters, trucks, houses, streets, bike pumps, toolboxes, hand-held vacuums, boxes, and robots in the city. There are also houses and each house can communicate with other houses. Each component in the game has a relevant computational concretization behind it.

In a conditional case, TookTalk starts and the user is flying in a helicopter over the city where finally he is landed in front of the house (Figure 7).

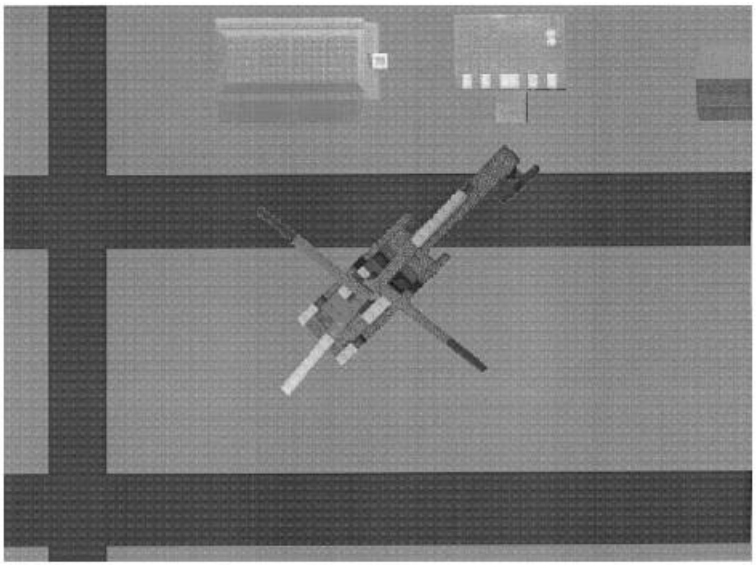

Figure 7: The player is flying over the city.

The user must search for the controls he needs in his inventory to build the necessary program. Whenever he finds the necessary tools, he picks up a robot from the inventory and starts training it with the tools he had previously selected (Figure 8). The user can train the robot until he finally produces a successful program.

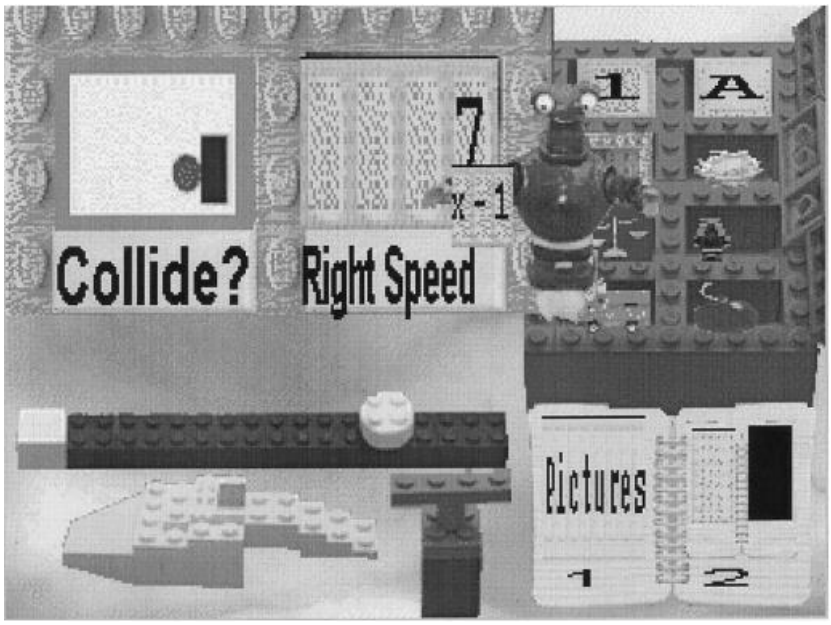

Figure 8: The player builds a program.

Box Island[4] is a block-based coding game that teaches the player an hour of code and is available on iOS and Android platforms. Box Island is a story-based coding adventure to introduce kids to the basics of computer programming. The player has to program Hiro, the avatar in Box Island, and move him around the island avoiding obstacles. Story-based adventure entertains and challenges the kids, or even adults while teaching the basics of coding logic. The adventurous game scenarios give a perfect opportunity for kids to keep their gaming concept alive and at the same time, they release the basics of coding along with this. There are overall 100 levels in the game, and 20 levels in the Hour of Code, a smaller version of the game (Figure 9) where overall each level is a puzzle that introduces the player to coding concepts like loops, conditions, etc.

[4] https://boxisland.io/

Figure 9: Box Island

CodeMonkey[5] is an educational game-based environment where kids learn to code without any prior experience. In CodeMonkey the player is launched on a magical island where a bad gorilla has stolen the bananas of the player's avatar. Across 100 levels, users learn to write simple code to guide their avatar, the monkey, to retrieve his stolen bananas. Each new level introduces an additional piece of code or a new function and challenges the player to put the pieces together. Since the player has solved the puzzle, the focus is shifted on getting each solution not only to work but to work well and efficiently so that they receive the maximum points and stars (Figure 10). The game challenges the players in understanding from simple to advanced coding concepts. There is also a version for kids aged 4-6 to get the fundamentals of coding.

[5] https://www.codemonkey.com/

Figure 10: CodeMonkey

A principled approach to teach programming is the debugging game called Gidget which uses a story to motivate the games' objectives [17]. In the storyline of the game, there is a chemical pill that is endangering animals and a robot named Gidget has been introduced to clean up the area. Though the robot is damaged, and the player is called to find the problem in the faulty code and fix it. The game is organized into levels and each level has missions that must be completed. Also, the missions serve and satisfy explicit goals and comprise assertions in the game's world. Gidget indirectly teaches programming concepts rather than a general-purpose language.

As the game is developed in a way that aids debugging, it offers controls that represent the functionality of conventional breakpoint debuggers in the code. The game simulates effectively the process of debugging offering a game experience similar to the real software debugging experience. Since the player completes the puzzle levels, he can proceed to the level of designer where he can create new levels and set the goals and the original code for them (Figure 11).

Figure 11: Gidget

In [18], the authors have developed four games including typing, multiple-choice, 3D action, and a programming learning game. All games are developed around the Java language and have different objectives. Firstly, the typing game “HIT Typing” is a simple game developed according to the concept of learning Java by typing samples of source code (Figure 12).

Figure 12: HIT Typing

The multiple-choice game, “Serious cube” allows the student to learn Java coding methods easily by using an Xbox360 controller as the operating device (Figure 13). The learning objective of this game is to allow the student to combine words used in Java and understand the language’s main structure.

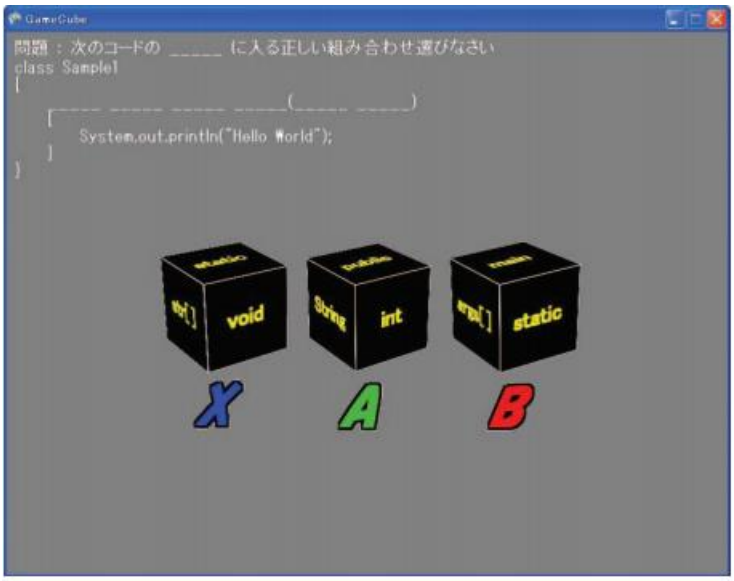

Figure 13: Serious cube

The third game that the authors developed is a 3D action game for 2 players and its learning objective is to engage the player in quickly grasping the missing words of the code to be completed and be executable (Figure 14).

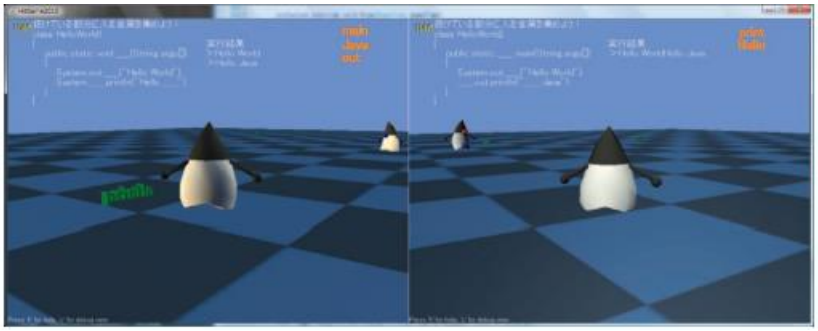

Figure 14: 3D action game

Finally, the programming learning game offers the learner the capability to control a character in the game and whether he wants to perform a certain action he can create a new command and add the control commands that he wants (Figure 15). Also, the game offers different stages and the authors note that the time to complete the stages reduces as the game progresses because each next stage's mission is more difficult than the mission of the previous stage. It is noteworthy that this game engages the player to notions like loops and conditionals, concepts that have been proven difficult for a novice developer to understand.

Figure 15: Programming learning game

Picobot[6] is a game created for an introductory course at Harvey Mudd College by Zach Dodds and Wynn Vonnegut. In the game, players write rules that instruct the green Picobot to navigate a specific grid. The blue cells cannot be traversed by the Picobot and are impenetrable as shown in Figure 16. The Picobot must navigate to the two-dimensional grid and turn all white cells into grey.

The game models a finite state machine. The player writes rules that dictate Picobot's next moves and simulate its navigation on the grid. The interface includes buttons that display the

[6] https://www.cs.hmc.edu/picobot/

current state of the program and facilitate debugging. The debugging as a learning process is also dominant in Gidget – the debugging-first puzzle game -.

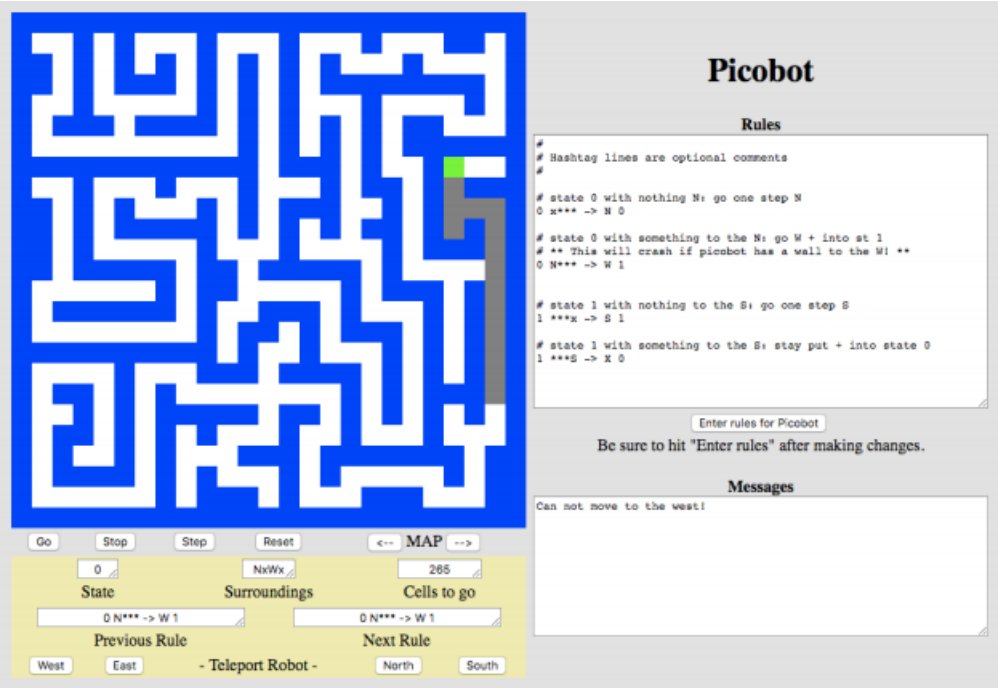

Figure 16: Picobot

In [10] the authors designed a prototype game where the player executes code using reduction-based operational semantics [19]. During the evaluation of the designed game, called Reduct, the authors recruited young adults with no prior Computer Science knowledge and found that after playing the game for a median time of about 34 minutes, participants could solve post-test problems around JavaScript with an average accuracy of 75%. Reduct was implemented in HTML5 and JavaScript ES2015 and was tested on Chrome and Safari browsers. The results of the evaluation indicated that gamifying the operational semantics of a programming language like JavaScript can be an engaging method of teaching programming concepts.

During gameplay, the player performs the steps of computation. Programming language theory offers several models of computation that break down execution into a recursive series of deterministic rules. The authors decided on using the rules of small-step operational semantics [19] in JavaScript, the most active programming language on GitHub.com[7].

Reduct contains 72 levels. In each level, the player sees three areas of gameplay: a board, a goal, and a toolbox. Figure 17 depicts a typical level that is relevant to the Conditionals section that players have to overcome. Also, the functionality of the toolbox limits the player's ability to brute-force the solution and the player is prompted to think of the solution before applying it to the interface. The expressions that the player applies to the toolbox are not sufficient to move him against or towards the solution or provide feedback on the solution.

To address the problem of self-handicapping [20], [21] the authors introduced the feature of Concreteness Fading (CF). While prior tools like Toontalk [16] have explored accessible concretizations of abstractions, these abstractions have not been faded to abstract forms over

[7] https://madnight.github.io/githut/#/pull_requests/2020/3

time. To evaluate the efficiency of concreteness fading the authors performed a between-subject design where half of the participants played the game using concreteness fading while the other half played the same game but with all visuals fully faded.

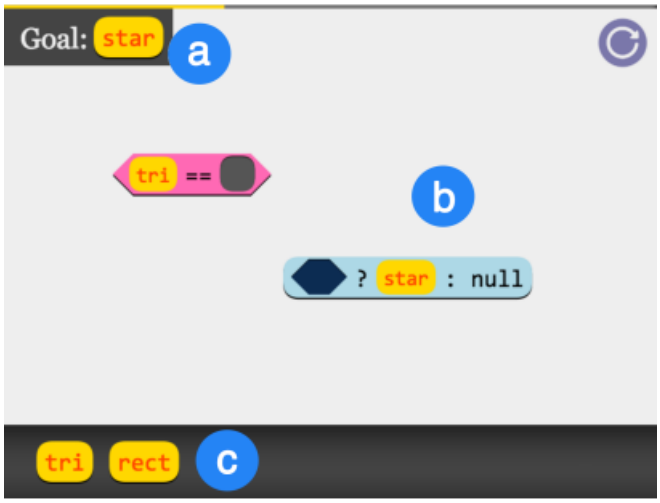

Figure 17: Reduct

Elevator Saga[8] was made by Magnus Wolffelt and its contributors. In Elevator Saga the player has to program elevators to minimize the number of times passengers are waiting to move from floor to floor. The game's goal is to give the player a lot of practice in JavaScript language. The player has to think creatively and experiment with different solutions and optimization approaches. Once someone starts the game, he is launched on the interface as depicted in Figure 18. There is a representation of the floors and the elevator's possible movements. He enters his code, written in JavaScript, below the game view and presses "Apply" to start the challenge. He must also declare an object that contains at least two functions called "init" and "update". The game not only helps the player understand and produce optimized code in JavaScript but also prompts the player to debug his program whenever he has not reached the final solution. Except for the movements of the elevators that require more complex and difficult code challenges in the following levels, there is no other storyline behind the game. Though, it provides analytical documentation and a help page where the user can find more information on the game's concept and the methods that can be supported programmatically for the elevator and the floor object.

[8] https://play.elevatorsaga.com/

Figure 18: Elevator Saga

Screeps[9] came out in 2014 and is a game that is still actively maintained. It consists of an MMO (Massively Multiplayer Online) sandbox strategy game for programmers. In Screeps the player can control his units, which live in the game even when he is offline, using real-life programming like JavaScript (Figure 19). The player can develop his colony, defend his borders, conquer new territory, and craft and trade items as in most MMO strategy games.

Figure 19: Screeps

Untrusted[10] is a browser-based puzzle game developed by Alex Nisnevich and Greg Shuflin. The game presents a rogue-like playing environment and a window with the JavaScript code you have to edit for each level (Figure 20). Once the game begins, the player finds himself trapped behind bars where he must code his way out. Below the window, the game also provides the player with a bunch of useful buttons and actions. One of them is the API that opens a reference window with the details of the functions that are supported in the game.

[9] https://screeps.com/
[10] https://alexnisevich.github.io/untrusted/

Untrusted is an extremely clever game and an open-source game in which someone can contribute to the project and develop a new level.

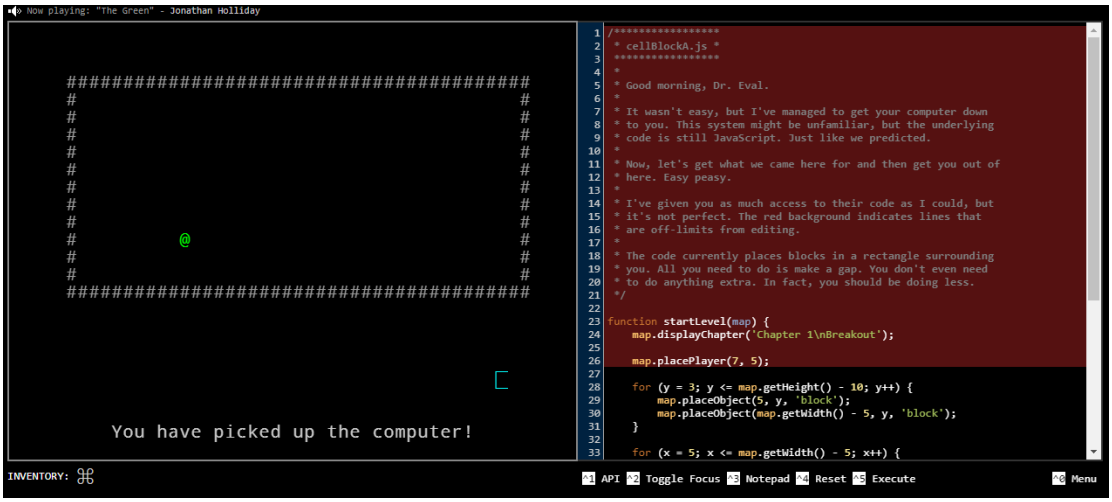

Figure 20: Untrusted

CheckiO[11] teaches the player TypeScript or Python in the style of Tower Defence-like thriller. CheckiO invites beginners and more advanced coders to a variety of programming challenges that are represented in the game via an archipelago full of islands (Figure 21).

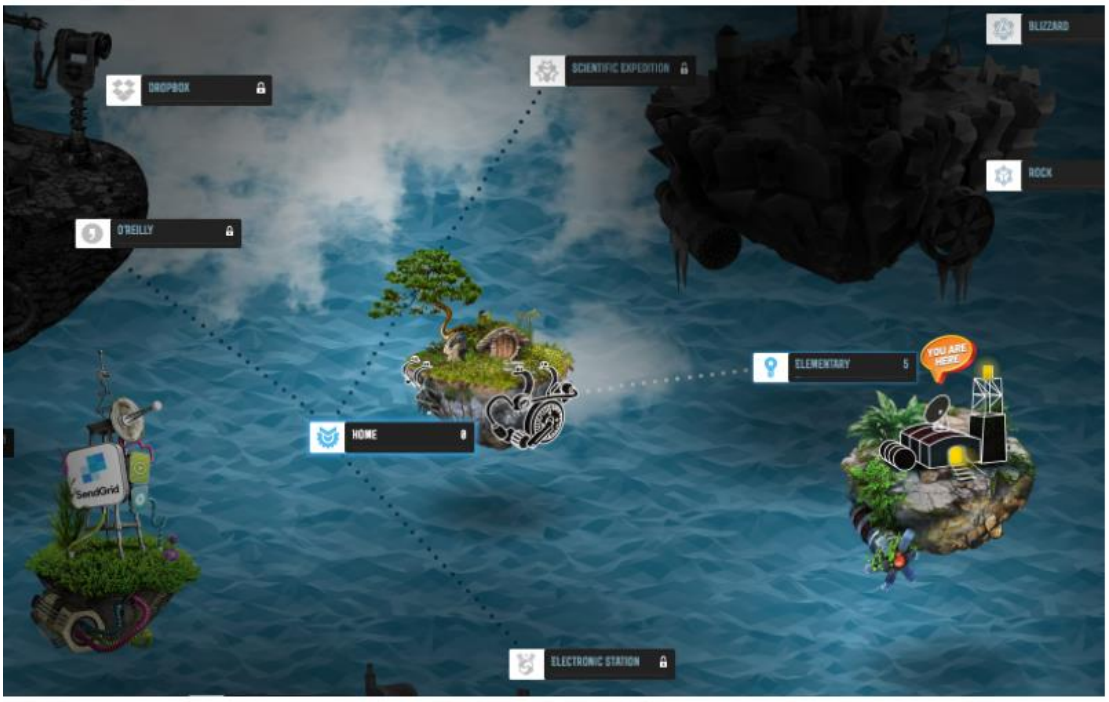

Figure 21: CheckiO

SQL Murder Mystery was created by Joon Park and Cathy He and adapted and produced for the web by Joe Germuska. Once someone starts SQL Murder Mystery[12], he receives the instructions and the information about the storyline behind the game. A crime has taken place in the game and the player's help is necessary to help the detective solve it. The player has to

[11] https://checkio.org/
[12] https://mystery.knightlab.com/

collect the information of the crime scene report he needs by retrieving data from the police department's database. Therefore, the player has to perform SQL queries to find the killer (Figure 22). The game helps the player understand and apply database concepts like primary and foreign keys, aggregate functions, etc. While moving forward in the game, the player needs to know how each table is structured and use his knowledge of the database schema and SQL commands to reveal the murderer.

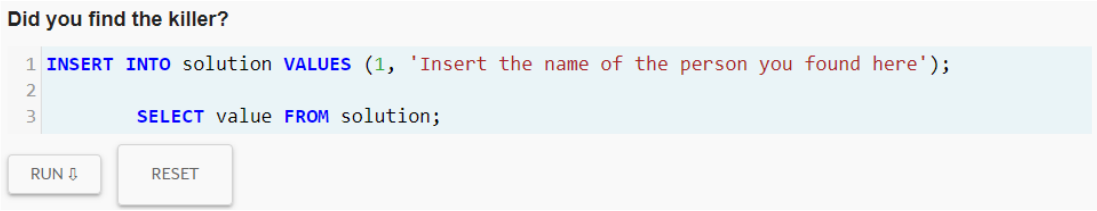

Figure 22: SQL Murder Mystery

Duskers[13] is a survival strategy horror roguelike game by Misfits Attic in which the player pilots drones through procedurally generated abandoned spaceships. The player is a programmer who is left alone in space with low supplies and has to deal with external alien threats that may sabotage his drones. The player throughout the game boards spaceships where enemies like automated ship defences, swarms of robots, or aliens lurk unseen and may disable his drones. He has also a grid-like map view of the ships which grows as the player explores more space.

The entire game is controlled by the keyboard by using the arrow keys for navigation or by typing into a command-line interface (Figure 23). The player navigates and programs the drones by combining multiple commands and providing the name of the drone and the room he wants to visit in a way that resembles shell scripting.

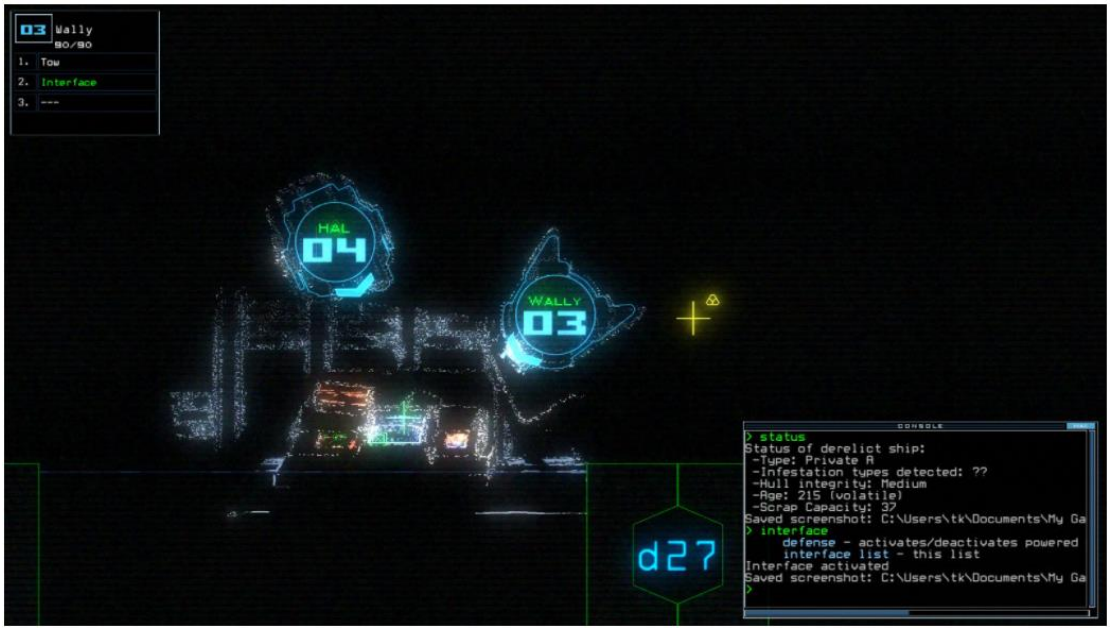

Figure 23: Duskers

[13] https://en.wikipedia.org/wiki/Duskers

In Super Markup Man[14] the player learns HTML and CSS in the context of a 2D puzzle platform. The game includes the basic HTML tags like ‘a’, ‘img’, ‘strong’, and ‘em’ and offers the capability of exporting the code in an HTML file. It teaches real web development practices without making the player write code and by simply carrying the tags around the interface and order them in the right way (Figure 24). Though, the game does not give explanations on the tags that it teaches making it difficult for novice players with no prior knowledge.

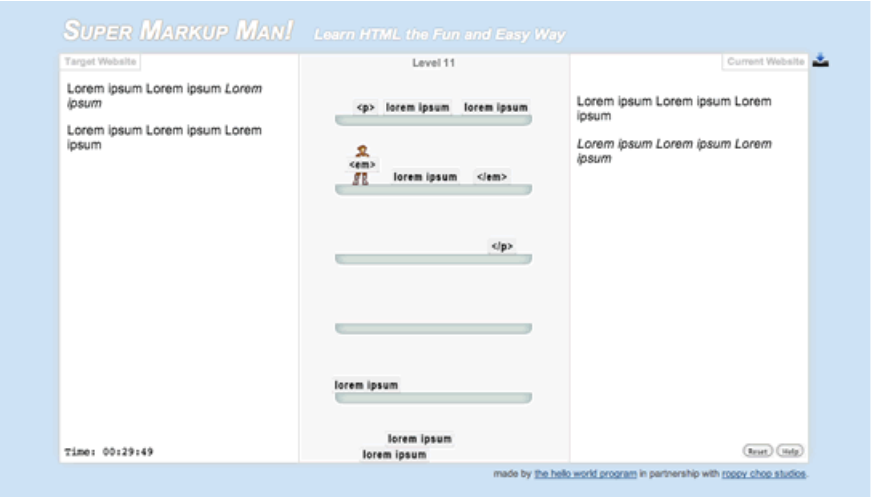

Figure 24: Super Markup Man

In Flexbox Defense[15] the player uses real-life CSS Flexbox code to defend his tower against invaders. The player has to position his towers correctly by using CSS (Figure 25) throughout 12 different levels. While entering the game a modal appears that provides the player with useful information about the flexbox component. The last levels are really fun and insightful and a little bit tricky for the player to find.

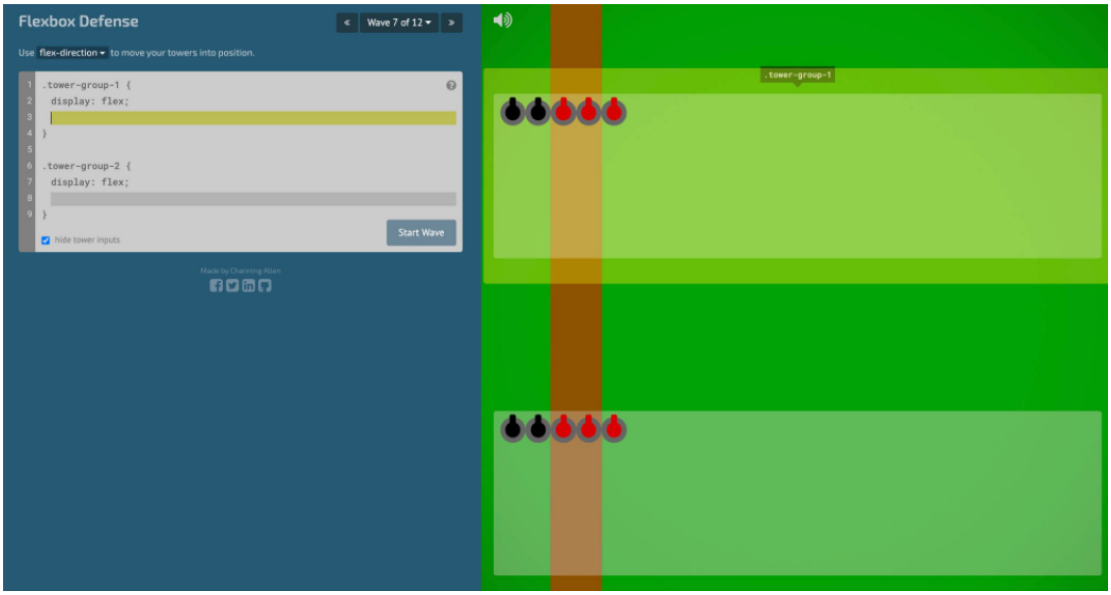

Figure 25: Flexbox Defense

[14] https://store.steampowered.com/app/502210/Super_Markup_Man/
[15] http://www.flexboxdefense.com/

In CSS Diner[16] the player learns how to use CSS selectors. He has to type the correct selector to complete each of the 32 levels. On the right, there is a help section (Figure 26) with relevant examples that guide the user indirectly to the solution. The player learns how to select a DOM element using a variety of different attributes like id, classname, nth-of-type, nth-child, not(x), [attribute="value"], etc.

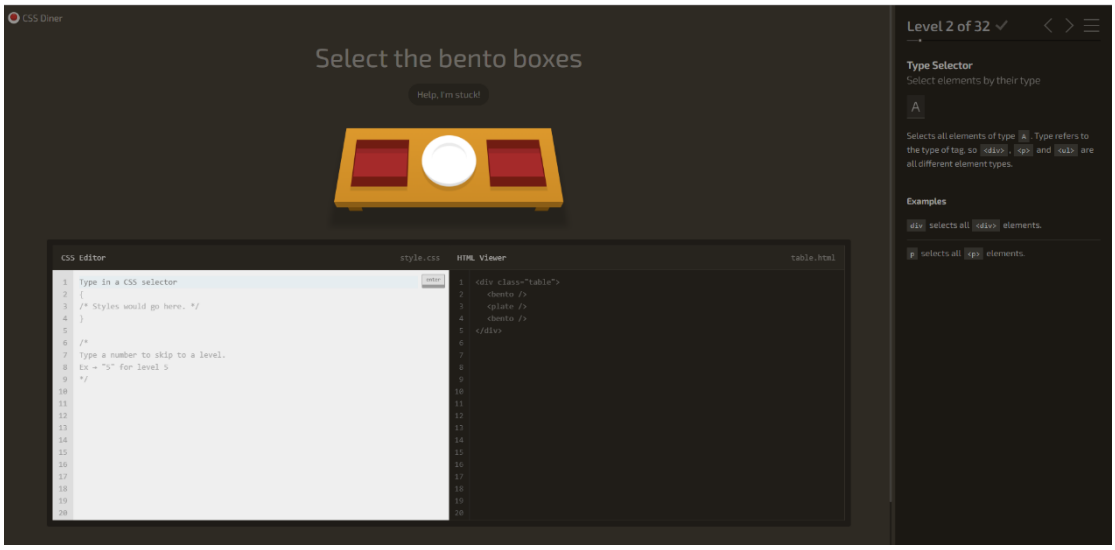

Figure 26: CSS Diner

Flexbox Froggy[17] is a game developed by Codepip[18] that focuses on CSS learning. During the game, the player has to help a frog and his friends find a lilypad displayed on the interface (Figure 27). The learning objectives of the game are to introduce the player to the fundamentals of using CSS flexbox to position elements on the game's interface.

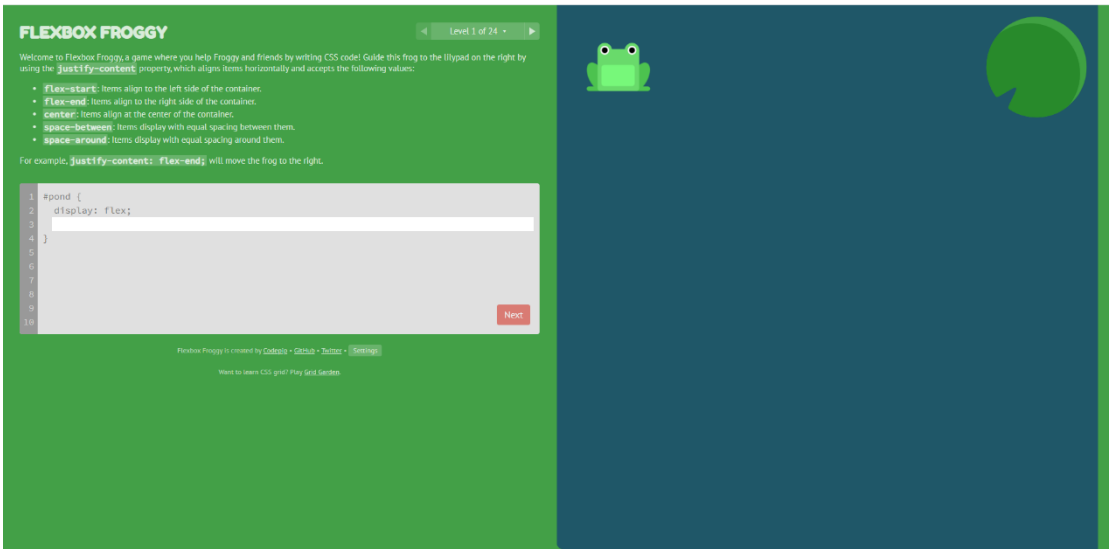

Figure 27: Flexbox Froggy

[16] https://flukeout.github.io/
[17] Htpps://flexboxfroggy.com/
[18] https://codepip.com/

In Grid Garden[19] the player learns CSS Grid Layouts by moving along the game that is divided into 28 different levels. The game covers properties like ‘grid-like-start’, ‘grid-column-area’, ‘grid-column’ etc. Like Flexbox Froggy the game is developed by Codepip and provides a very similar user interface. The storyline behind the game motivates the player to grow a carrot garden where he has to use specific properties relevant to Grid CSS and water certain areas on that grid (Figure 28).

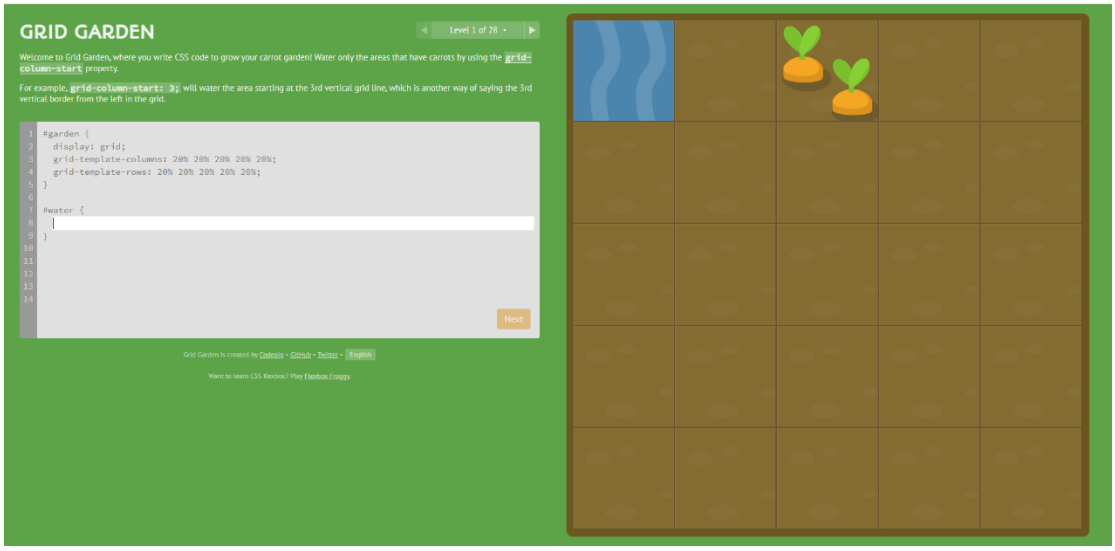

Figure 28: Grid Garden

Flexbox Zombies[20] is an educational game where the player fights zombies by using features of flexbox to aim his crossbow towards the zombies (Figure 29). The game teaches flexbox through a story and reinforces the fundamentals of flexbox in a fun and effective way.

[19] https://cssgridgarden.com/
[20] https://geddski.teachable.com/p/flexbox-zombies

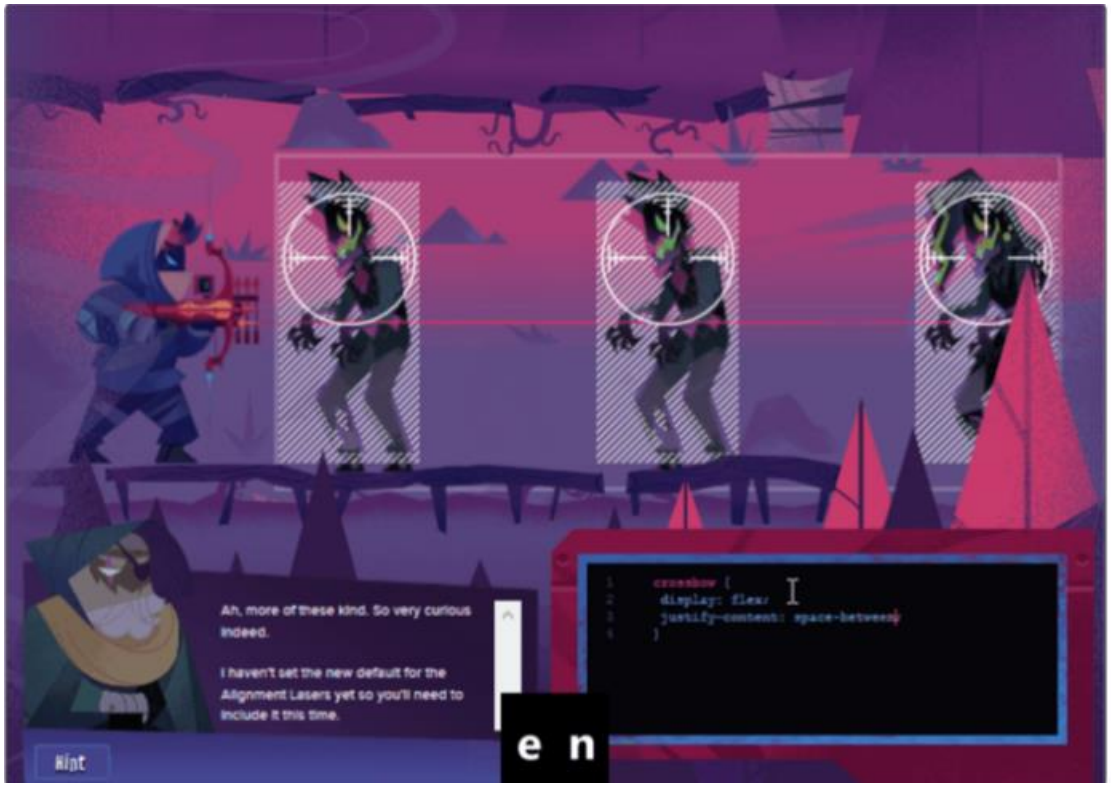

Figure 29: Flexbox Zombies

CodinGame[21] is a collection of several minigames in which the player can code nearly in any language he likes. CodinGame (Figure 30) offers a new way for developers to improve their skills and have fun at the same time. Also, he can discover new languages, algorithms, and tricks in courses crafted by top developers. In CodinGame platform, popular programming languages like Java, Python, Kotlin, Scala, JavaScript, Swift, Ruby, Rust, C#, C++, C, Groovy, etc. are supported. The player can work on programming puzzles that are in varying skill levels from easy to hard.

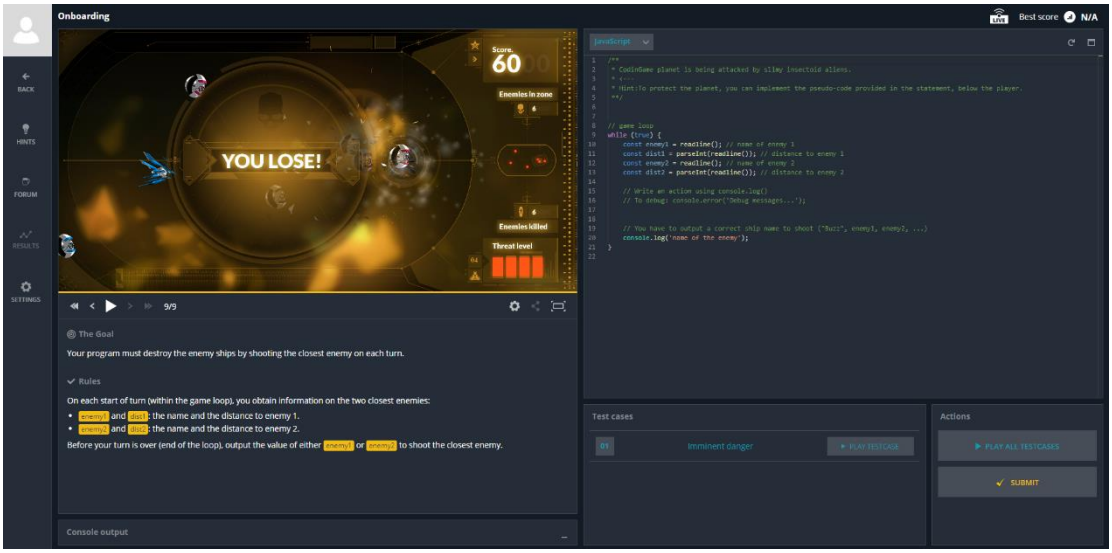

Figure 30: CodinGame

[21] https://www.codingame.com/

Except for the programming languages, editors are an essential tool for software developers. In VIM Adventures[22] a player can learn VIM keyboard shortcuts and essential VIM concepts without being frustrated that he does not remember all commands he has to use. The player can discover new tricks and commands in the editor and see the full documentation of the game and VIM capabilities by typing help on the command line. Below the interface of the game (Figure 31) some buttons offer actions on the content of the game or the game itself like time, keystrokes, etc. There is also the keyboard button where a novice user can see all the functionalities that are hidden behind each key. Whether he needs more information he can also type the relevant command on the command line.

Figure 31: VIM Adventures

Swift Playgrounds[23] engages players in the world of the Apple-specific Swift programming language and is available only on iOS. In the game, the player can conquer levels, puzzles, and coding concepts. The player starts with the "Fundamentals of Swift" lesson where he reaches goals using the same code that professional developers use every day in Swift. As the player moves along the game he is introduced to more advanced concepts. The game is overall delightful and as depicted in Figure 32 the game follows the discipline of "what you see is what you code". The user creates code on the left side of the screen and instantly he can see the results of the code on the right side.

[22] https://vim-adventures.com/
[23] https://www.apple.com/swift/playgrounds/

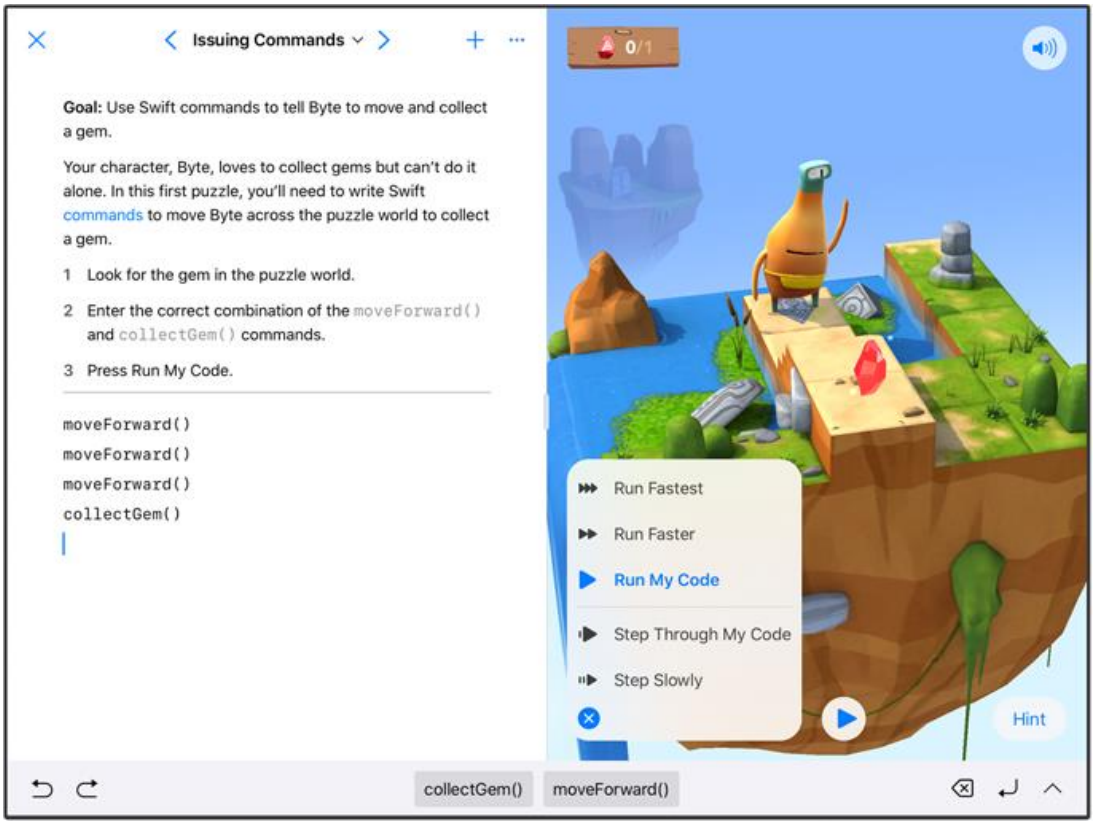

Figure 32: Swift Playgrounds

2.6 Review of Case studies

The authors in [17] propose that debugging games can be broadly accepted and learning environments can benefit from adopting this debugging-first approach. Also, Gidget provides promising results for teaching students how to use loops and functions.

The general orientation around the knowledge that should be acquired while playing a serious game about software development focuses on what programs are for and what a user can do with them. The player of a serious game of this kind should efficiently learn how to plan, develop, test, and debug a program. So, except for the development, debugging is a dominant activity of a software developer and this fact resonates with the nature of games like Gidget [17] that deploy debugging as a valuable method for learning. We also note that debugging also exists in Picobot and helps the user find out the state of the program at each iteration.

New learning environments also include Virtual Worlds (VWs) and like serious games, they emphasize interactivity and immersion. There are also Living Worlds (LVs) which are Virtual Worlds populated with Non-Player Characters (NPCs) that perform goal-oriented activities and interact with the player [26]. In these environments, the player is a Virtual Human Character (VHC) and controls an avatar in the game.

2.7 Learning Outcome

With coding puzzles that are naturally scaffolded into increasingly difficult challenges, serious games are a great way for the learners to start thinking computationally and critically. Also, many games encourage trial-and-error techniques and creativity while at the same time learners obtain technical knowledge around programming.

The software development process is involved with the development of mental abilities and the progressive way in which students learn to think computationally. Research on Bloom's Taxonomy of Learning [22] suggests that students learn in a hierarchical manner [23].

Most researchers conceptualize learning as a multidimensional construct. While evaluating a serious game for software development as an instructional method, we should also evaluate the learning outcomes of the games. As we study serious games, it is a fact that this type of educational game can produce learning and psychological effects. These effects may be intended or unintended [5]. There are several broad categories of learning outcomes: skill-based, cognitive, and affective outcomes [24], [25]. The cognitive outcome is related to the fact that students have the opportunity to learn more deeply. Affective outcomes cluster several subjective phenomena like acceptance, emotions experienced, motivation, transversal skills or competencies, and behaviour change [5]. As for emotions that players experience, though it is an important factor, there is no universally accepted classification of emotions. Also, in games, we can notice undesirable behaviours like cheating or hacking. Other effects may involve motor skills, perceptual and cognitive effects, and psychological effects.

3 Game Design

An educational method for learning to program while playing a serious game that aims at software developers with varying prior knowledge of programming is proposed. In this study different types of games and mechanics are developed and evaluated based mostly on a Human-Computer-Interaction (HCI) aspect. HCI defines whether the designed system is functional to the users and stakeholders and by what means users maximize their benefits and achieve their expectations. Since serious games for software developers require constant interaction to evaluate and further improve their abilities, the mechanics in this interaction and overall usability are the key aspect of design. Also, another aspect that is evaluated is the performance of software developers in not only the production of the code but also the debugging of a faulty code. As previously discussed, identifying and implementing possible solutions are required to be an accomplished problem-solver and programmer. Two ideas follow different activities, production and debugging. Debugging as a process involves combinatorial logic and should go beyond the task of finding the flaws in the code. The main challenge in debugging is to divide the code into parts and identify the specific location of the problem. On the other hand, debugging provides the developer with the code, even it does not work perfectly, while in production mode the developer finds all the solutions from scratch. All the above parameters are introduced in the implementation of the game and they are further analyzed and discussed.

3.1 Conceptualization

Game design is the process of designing a game and game development is the process of taking the design and implementing it. The researchers should know the game genre that they are interested in so that they study what elements they could improve and do better. Also, except for the game genre, designers should think of the players and how to properly balance the game so that they provide them with as much challenge as they want without making it too easy or too hard. There are several aspects through the development process that a researcher can find useful to finally produce a more enhanced game.

Firstly, mood boards are a great representation of the art style, typography, and early visual design of the game and simulate the look and feel of the conceptualized idea. In Figure 33, a mood board is depicted that was specifically designed for the conceptualization of the game. The mood board consists of a planned arrangement of images, materials, memes, and pieces of text that communicate the style of the game. The goal of the mood board is not to describe in detail either the game or the characters, though it sets its direction and an overall context. The material is organized in a way that combines at the same time colours, textures, environments (fictional and urban), memes about software developers and gaming, snippets of code, avatars, and design patterns.

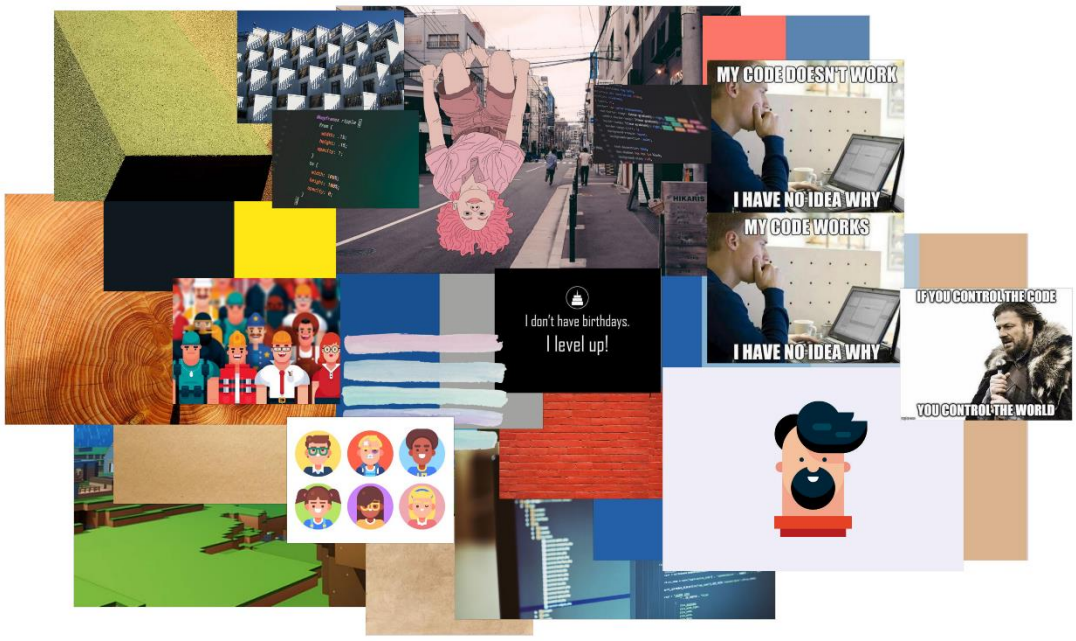

Figure 33: Moodboard for the conceptualization of the game

Secondly, storyboards provide a visual representation of the gameplay and the narrative behind the game. In Figure 34 the scenes of the game are depicted along with captions that give explanations about each scene. The player is placed in a city where several challenges arise. To solve the puzzles and reach the target, the player has to produce the code that moves the avatar towards the prize. If the avatar reaches the prize successfully, the player completes the level and moves into another challenge.

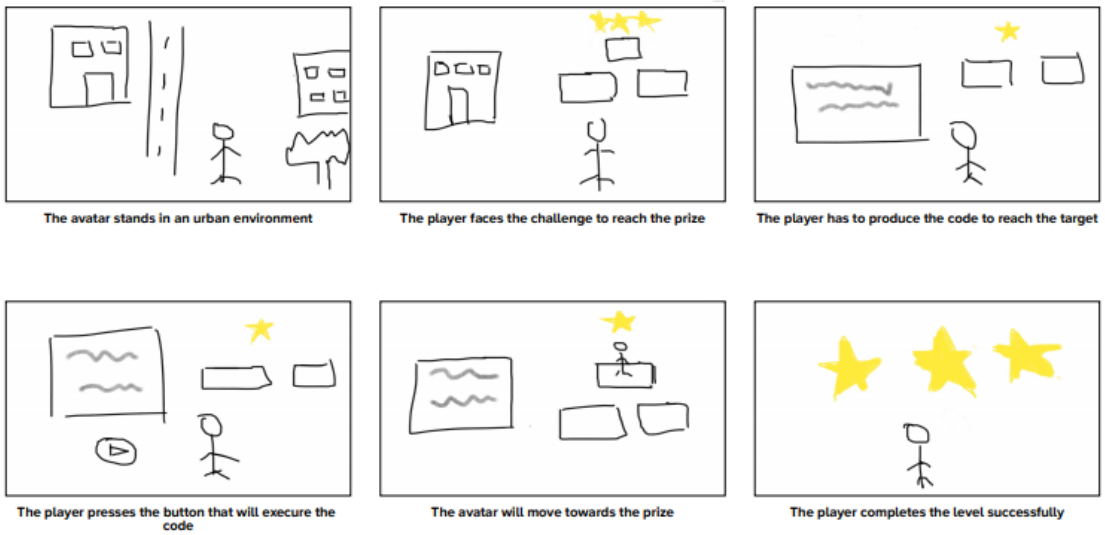

Figure 34: Storyboard

3.2 Game Design

Since this study explores different types of games and mechanics, four mini-games were developed based on the concepts that are already discussed. The main idea is splitting the game into different versions of the same content taking into consideration two parameters: the type of interaction and the activities, either debugging or production of the code. The first one represents the way that the user interacts with an interface whose main task is the

production of code. While thinking of the different types of interaction that a software developer is familiar with, two basic affordances arise. An interface where the software developer uses exclusively the keyboard and is focused on typing interaction generally and an interface where the developer uses exclusively the mouse. So, in this category and following the idea of comparing and evaluating different interfaces in a serious game, the options of typing and drag-and-drop were dominant. The second parameter defines whether the user will either produce the code or debug a faulty code that is already provided in the interface.

Finally, the game consists of the four following versions of the same content:

1. Drag-and-drop interface – production mode
2. Drag-and-drop interface – debugging mode
3. Typing interface – production mode
4. Typing interface – debugging mode

Since all game ideas were defined, producing the design through prototyping was the next step of the development process. Prototyping is a very crucial phase of the early development phase and it is often neglected by the developers. While prototyping there is not much time invested in the code architecture and even though paper prototyping can show some advancement of the game it does not necessarily show all its qualities. Paper prototyping in Figure 35 was initially developed to determine whether the game's flow is effective, and players are learning throughout the game.

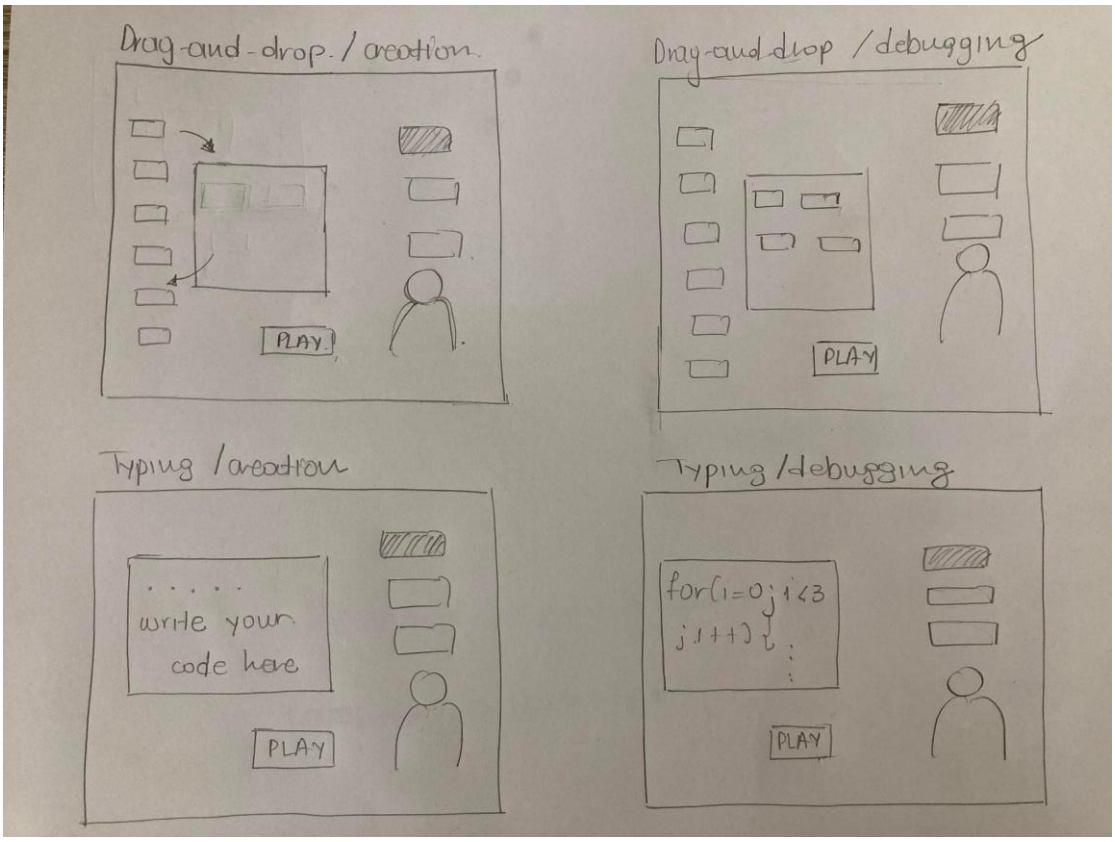

Figure 35: Paper Prototyping

Since paper prototyping shows some basic aspects of the game, digital prototyping was implemented in Figma[24] for further advancement of the game's flow. This step was proved very crucial for the final implementation of the game as major issues were corrected before starting the implementation.

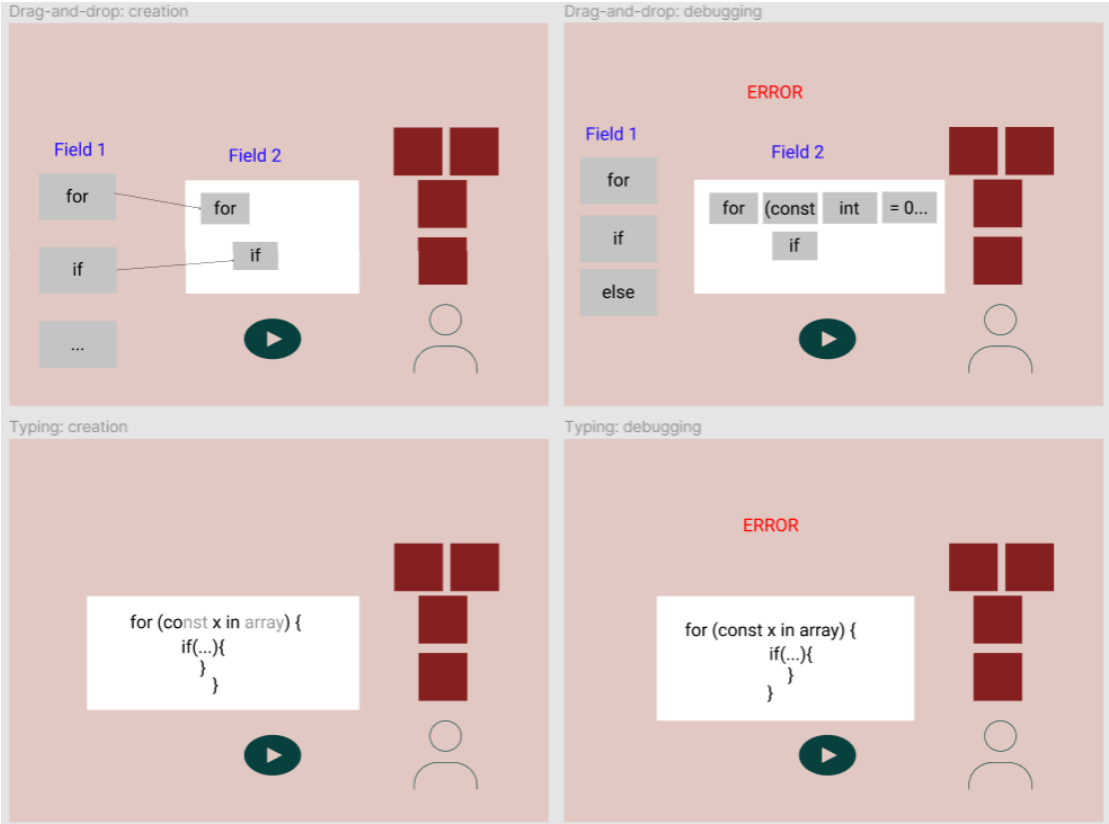

Figure 36: Prototyping in Figma

In Figure 36 the frames of all interfaces are depicted without divining in detailed representations of the environment or the avatar of the game. While analyzing the above frames, a major concern was developed for the typing interface and the necessary information that is available to the player. The two interfaces, drag-and-drop and typing, are inconsistent as for the data they provide to the user leading to the conclusion that the drag-and-drop interface seems to be more helpful as it already contains the commands that the player will use. Considering the above assumption and to balance the level of difficulty between the interfaces, a variation of the typing game was proposed. While typing the commands of the game will appear to the player by using the autocomplete method that already consists of a built-in feature of most used code editors. So, the player is provided with the commands in real-time and pressing a single key on the keyboard will complete the command without typing all the letters. This method was also selected as it contains the ability to type and produce code faster and many software developers are familiar with it.

3.3 Research Questions

The goals of this study include the following research questions:

1. **RQ1**: Are there any differences in effectiveness related to the type of interface?

[24] https://www.figma.com/

To examine the effectiveness related to the type of interface, we examine the number of times that the participant provided a wrong answer to the system. All cases were examined to study which interface is overall more effective. Also, as we have already divided the activities that the player performs into two categories, we study which interface is more effective in activities where the player creates or debugs the code. To measure the differences in interface's effectiveness we state the Null and the Alternative Hypothesis and examine whether the means of the two samples are different performing a two-tailed test. So, our hypotheses are the following:

$H_o: \mu_1 = \mu_2$
$H_1: \mu_1 \neq \mu_2$

2. **RQ2**: Are there any differences in efficiency related to the type of interface?
   We measure as efficiency the time that participants needed to complete the game. As in RQ1, we state the same Null and Alternative Hypothesis and we test the overall efficiency including both cases that participants created or debugged the code. Also, we examine whether the drag-and-drop or the typing interface is more efficient while players create or debug the code.
3. **RQ3**: Are there any dependencies between the interface and the failure rate?
   We test whether the interface can be linked to the failure rate of participants to complete the game.
4. **RQ4**: Are there any dependencies between the interface and usability?
   We examine the usability of the interfaces by using the questionnaire that participants answered and test the questionnaire's reliability to decide whether the scale is reliable.

3.4 Decision of game engine

Game engines that create interactive games have existed for decades and many disruptive changes have been made in that field. Over time, engines that support 3D games were published opening the way to the initiation of new members into the community and game development overall. These engines, mainly nowadays, support the play of games in nearly all digital available devices like smartphones, game consoles, laptops, VR platforms, etc.

Unity[25] is a cross-platform game engine that has obtained great achievements and is at the moment a leading platform in creating 2D, 3D, and VR games, and also the platform that is used for the development of the game of this study. The game environment in Unity consists of a world space with x, y, and z-axis for 3D games and only x and y for 2D games. Anything that can be part of the game like the avatar, objects, camera, etc. is called Game Object and is placed on the axes. Complementary to the Game Objects, components, like Transform, Sprite Renderer, Colliders, etc. that define its behaviour can be attached to it. Transform component defines an object's position, rotation, and scale concerning the world. Sprite Renderers provide the Game Object with a real shape in a 2D or 3D world so that the player will be eventually able to see it. Colliders, as defined by their name detect the detection of a collision between Game Objects. Except for the behaviour of predefined components, the developer can add custom components in the game as scripts and describe the functionality that the Game Object will perform via this custom component. Scripts are mainly written in C#, an object-oriented language.

[25] https://unity.com/

Except for Unity, another popular game engine is Unreal Engine[26]. Unity and Unreal Engine present many similarities but at the same time many differences as well. Some of the differences that lead the developer to the final decision are relevant to the programming language that is used for scripting, processing, and rendering supported libraries and tools, etc. Firstly, as beforementioned Unity uses C#, a simpler programming language than C++ which is used in Unreal Engine and requires a deeper understanding of its behaviour. Though, Unreal Engine supports Blueprints, a flexible and powerful system of visual scripting that uses a node-based interface where the developer produces the connections among the gameplay elements. As for processing and rendering Unreal offers a more robust background but at the same time requires the developers' competence in graphics. Unity also offers a vast majority of tools for 2D, 3D, AR, and VR games without forcing the game to become slow or dysfunctional.

Various reasons contributed to the decision of implementing the game of this study in Unity. Firstly, Unity is not only a great and non-complex environment for a developer to start with game development but also supports all aforementioned devices. Secondly, the developer uses programming languages like JavaScript and C# that enrich the game's development. Also, the platform is distributed with an open-source license and is fully supported with a massive collection of libraries, tools and assets. Finally, Unity is supported by a large community through forums and online tutorials making development and debugging more convenient and less stressful for an independent developer.

The game of this study is developed in 2021.1.7f1 Unity's version and built by WebGL platform and all scripts were written in C# programming language.

3.5 Unity and C#

In this chapter a brief description of properties and functionalities, that consist of the necessary information for the development of a game, of both C# and Unity is presented. Especially Unity C# is standard C# that works with Unity's components. While looking into some general aspects of this programming language, it is useful to get a deeper understanding of how a game in Unity can be implemented and developed through scripting.

3.5.1 C#

C# is an object-oriented programming language, developed by Microsoft and based on Java, a general-purpose programming language. C# and Java are similar languages and share common properties, though there are some differences between them. In Java, as for classes, there is only a single public class while in C# there are multiple public classes in the source code. Also, structures and unions are supported in C# while in Java are not.

Like many other programming languages, C# has a garbage collector, in other words, a thread that runs concurrently with the program's execution and releases memory that has been allocated and is not used anymore by the application. As for Unity, this feature constitutes a great advantage since it releases the developers from the responsibility of performing manually this task like in C++.

Inheritance in C# has the same functionality as any other object-oriented programming language and polymorphism is also supported. Inheritance is the process of acquiring all the properties of one class into another class. There are two classes, the base, and the derived

[26] https://www.unrealengine.com/

class. The base class is known as the base class whose properties and methods will be inherited to another class. The derived class is known as the child class and has the properties and the methods of the base class. Inheritance helps in achieving the polymorphism that allows an object to represent more than one type.

Variables in C# are divided into two categories, reference types, and value types. A value type holds the data within its memory allocation and a reference type contains a pointer to another memory location that holds the data. The reference data types do not contain the actual data stored in the variable, but they contain a reference to the variable. If the data is changed, this change is also reflected in a change in the data of the other variable. Reference type variables are stored in the heap while value type variables are stored in the stack. The built-in reference types in C# include object, string, and dynamic. Examples of reference types include strings, arrays, objects of classes, etc. On the other hand, value types include boolean, date, structs, and enums.

Finally, a delegate in C# is a reference type variable that holds the reference to a method. The reference can be changed at runtime. Delegates are used for implementing events and call-back methods. Also, lambda expressions in C# are used like anonymous functions, with the difference that in lambda expressions the type of the value is not required to be explicitly specified.

3.5.2 Unity

Unity provides the developer with a library of C# and many functionalities that enhance the development, design, debugging, and building of the final project. A brief analysis of the main features of Unity is presented below, though the full documentation is also available online[27].

Unity supports not only the classes' hierarchy but also the main class from which all incorporated classes can inherit. This class is called Object (having the same name as the main class in C#) and constitutes the base class for all built-in objects in Unity. All objects can refer to this class (UnityEngine.Object) and although Object is a class it is not intended to be used widely in the script.

Moreover, Unity Editor is the core editor in Unity whereas a game can be developed. In Unity Editor all actions of game development can be supported except for scripting and production of image and audio files, etc. Except for the assignment of scripts, audio, images, or other types of files to GameObjects, the developer can create Scenes where GameObjects are included. Moreover, the developer can define variables of GameObjects like transform, rotation, scale, and many other features as well. The processing of data like strings or variables that will be used in the game is also available.

As beforementioned, in Unity, a game can be organized in Scenes, where GameObjects are placed. A separate window, where active Scenes are displayed, is provided so the developer has the opportunity to preview the structure of the scene that will be loaded during the program's execution. Unity supports also parenting. To make any object parent or child of another GameObject, the developer drags the desired child onto the desired parent in the Hierarchy. The child will inherit from now on the movement and rotation of its parent. Another helpful enough feature of Unity Editor is Play Mode. Play Mode supports the

[27] https://docs.unity3d.com/Manual/index.html

simulation of the game and its execution in its final form. Unity auto-saves the scenes every time play mode is enabled and saves these scenes in a folder.

Programming in Unity usually includes the process of writing scripts that behave as custom, editable components that the developer can attach to GameObjects. In scripting, some common methods are Start, Awake, Update, FixedUpdate, OnEnable, OnDisable, OnGUI, etc. and Unity calls these methods in scripts during each frame.

The variables of GameObjects and their components are serialized, though some requirements must be fulfilled so that a variable can be legitimately serialized. Valid types include primitive types like int, float, bool, string, etc., the UnityEngine.Object and all classes that inherit from it, some structs of Unity like Vector2, Vector3, Color, etc. Also, non-abstract classes and structs can be serialized since System.Serializable attribute is assigned to them. Finally, a problem arises when the developer needs to change the name of a variable that has been serialized. Serialization uses the name of the variable to save her value, so if changed her name will be received as a new variable and the already stored value will be lost.

Unity also provides layers to the developer and GameObjects can be assigned to them via the Inspector. Layers in Unity help indicate functionalities across GameObjects such as which GameObjects will be ignored by raycasts or be invisible to the camera. This system is useful for building complex systems or aiding in functionalities like collision detection.

4 Game Implementation

This chapter includes the entire process of the game development of all four variations of the game. Firstly, the type, the purpose, and the decisions that were made on the game are analyzed in detail. Also, the mechanics and the user's interface are explained and finally, the different versions and levels are described.

4.1 Game's overview

The game of this study is a computer, serious game that requires the player to code to finish it successfully. The game addresses software developers or users with prior knowledge of coding and it is not intended to teach programming from scratch. The main purpose is based on the idea of promoting further advancement of programming skills and computational thinking that software developers cope with. So, this game is an approach to the integration of educational methods into the game to not only teach computer science principles but also promote the engagement of developers to problem-solving activities through game-based learning. This study aims at improving the ability of the developer to learn and practice coding outside of the traditional context and experiment with different concepts and procedures.

Players in the game are located in a learning environment where they face puzzle challenges in sequence. At this point, it is interesting to tackle the issues that arise while designing the content of the challenges the player faces. Firstly, the game addresses players with varying academic or professional backgrounds in coding. So, the notions the game includes were selected to be the ones that are not only common between the players but also most difficult to understand. The game focuses on problems that novice programmers and developers struggle to become familiar with, though they come in contact with them through an educational and enjoyable experience.

The game is a 3D, puzzle game where the player is located in a city and four different versions have been designed which share common mechanics, puzzle challenges, and levels. Briefly, the versions are the following: drag-and-drop interface and production's code mode, drag-and-drop and debugging mode, typing with autocomplete and production's code mode, and finally typing with autocomplete and debugging mode.

All versions of the game are organized in levels, and more specifically each mini-game consists of 5 levels and each level includes the player, represented with an avatar, located at a different place.

4.2 Environment / Player / Other components

The game is taking place in a city and the player is represented by an avatar in the gameplay. The player controls the avatar of the game by writing the code that solves the puzzle of each level moving the avatar towards a specific and well-defined route. Once the player has successfully produced the final code, the prize is reached by the avatar and the level has been completed.

4.2.1 Game Environment

The levels of the game generally take place in a city and each level is located at a different spot. These variations in the game environment make the gameplay more interesting and keep the player's enthusiasm for the game. To follow the idea of incorporating an urban environment in the game and allow the player immerse more into the game, the integration of a Unity asset in the game became more prominent. Since Unity's Asset Store is full of not

only high-quality implementations but also assets that can be very easily adopted in a Unity project, it was not difficult to find an asset to start with. Polygon City Pack[28] seemed to satisfy all requirements that were set during design as it offers a beautiful pack of models. This asset depicts an entire city, fully designed with all relevant objects developed and gives the player the idea of moving in a large and continuous space. So, all scenes as well as the main game environment were based on the asset of Polygon City Pack as in Figure 37.

Figure 37: Game environment

The game offers a diversity in the visual scenery and a realistic representation of a city of a virtual environment that consists of trees, fences, stones, grass, etc. The environment includes exclusively prefabs, predefined GameObjects that have been developed and can be reused in the game. These prefabs have specific components attached to them that define their properties and their behaviour. The main feature of the environment's implementation is that it is fully developed by prefabs and can be customized easily.

Since the asset of Polygon City Pack offers many different locations and scenes, the avatar was selected to be always located at public open space keeping surrounding as simple as possible as in Figure 38. The decision of not having too many details in the environment aims at keeping the player focused on the puzzle and undistracted from superfluous information.

[28] https://assetstore.unity.com/packages/3d/polygon-city-pack-environment-and-interior-free-101685

Figure 38: The avatar in the city

4.2.2 Player / Avatar

The player is a character in the game and at the same time is also the only character in the game. The game does not support non-playing characters and the avatar cannot have any other interaction or conversation in the game. For the avatar's representation, an asset with Human Characters was used[29] and the avatar's appearance was fixed as depicted in Figure 39 and not able to be selected or customized by the player.

Also, the player can move the avatar only since the correct answer is given to the game and not by clicking on the interface or by using any key on the keyboard. So, the avatar does not move randomly and even if the player gives an incorrect answer, it does not move at all. Though, since the player has successfully entered the input that moves the avatar towards the prize, it starts moving gradually towards the prize. The avatar can move forward, back, right, and left and turn right and left.

Figure 39: The avatar

[29] https://assetstore.unity.com/packages/3d/characters/human-characters-free-sample-pack-181554

4.2.3 Other components

To move to the next level in the game the player is required to solve a puzzle and move the avatar towards a path. So, in each level except for the floor that is already depicted as part of the city's environment and may include materials and objects like grass, street, pavement, etc. the path the player should follow consists of certain tiles. The tiles were based on Unity's asset in asset store which is called HYPEPOLY – Isometric Tiles Standard Lite[30] and the materials of the tiles were customized to be visible on the ground (Figure 40).

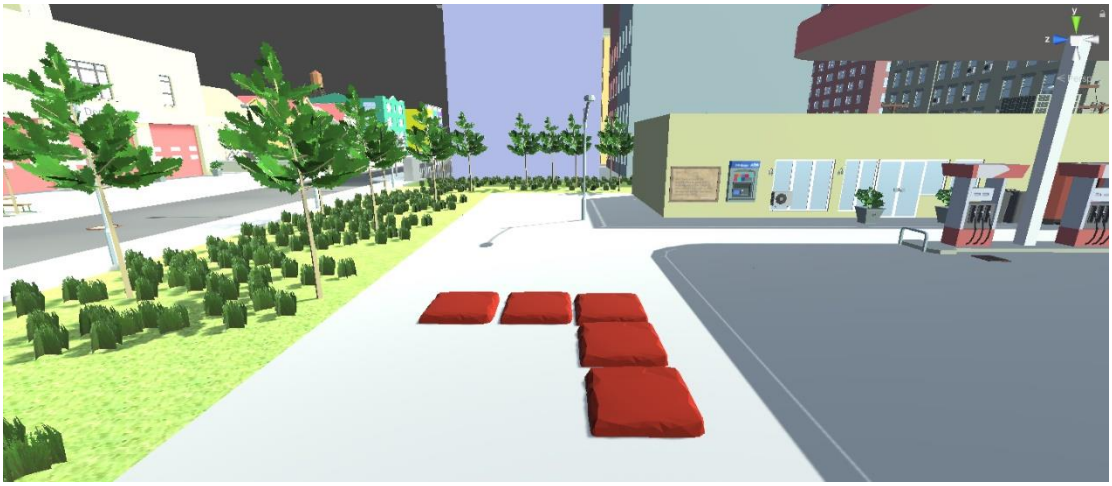

Figure 40: Tiles

The player moves only on the specific red tiles on the floor following the route towards the prize as in Figure 41. The object of a gold cup was selected to represent the prize and the final target of the player's path as it is an object that is widely defined as a user's goal in most games. So, it is considered that the player would be determined to plan the path towards the only object placed above the final tile, except for the semantics of the gold cup. For the gold cup in the game, an asset called Trophy Cups/Chalices[31] is used for the realistic appearance of the cup.

Figure 41: Prize

[30] https://assetstore.unity.com/packages/3d/environments/hypepoly-isometric-tiles-standart-lite-193779

[31] https://assetstore.unity.com/packages/3d/props/trophy-cups-chalices-free-188059

One of the most significant components in the game that should not be neglected is the Camera Component of Unity. The Camera supports the visual representation of all objects during gameplay and as displayed in Figure 42 it is a third-person camera located at the shoulder level of the player. The position of the camera offers an overview of the scenery and provides the player with visual contact with the avatar and the puzzle at the same time.

Figure 42: Camera above the player

4.3 Game Mechanics

In this chapter, the main mechanics that are used in the game will be described. All of them are used in all versions of the mini-games following the same rules to keep the consistency between them. Briefly, the mechanics and tools that will be analyzed are the levels, the timer, the score, the "Help" button that provides general instructions, the "Hint" button that provides the player with a piece of information on how to solve the puzzle and the "Clear" button that simply resets the player's input.

4.3.1 Levels

Before the player is launched to the actual game, the main menu was designed to introduce the player to the game (Figure 43). This transition gives the player the time to adjust smoothly and makes the player's transition to the game less rough. During this start scene, the player can take the time to read the instructions provided before feeling ready to move to the game.

Figure 43: Start Menu

Each mini-game is organized into levels and all min - games include five levels. Each level requires the player to combine all prior knowledge from previous levels and use acquired skills to solve the puzzle. So, the first level starts with a simple task and the player is moving gradually to more complicated tasks until the final fifth level.

Since the player completes the level and solves the puzzle, a pop-up message is displayed with a message. As displayed in Figure 44 the player is informed that the answer that was provided is successful and is directly moved to the next level. Since the player clicks on the "OK" button, the next challenge is automatically loaded. It is also important to encourage the player for this achievement and amplify the player's confidence during the game.

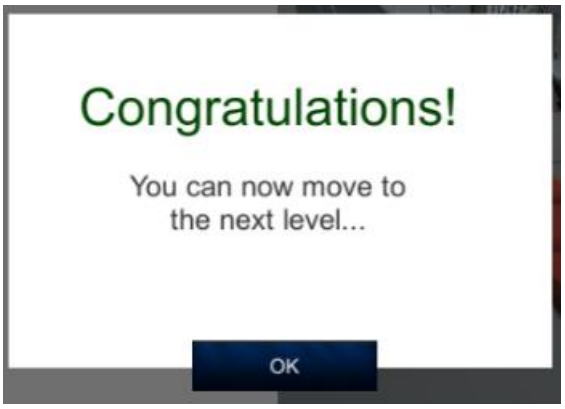

Figure 44: Successful message

Finally, levels are organized and interpreted as scenes in Unity's environment. Each scene includes the player's state of the previous scene including the score which is accumulated during the levels and it is finally calculated once the fifth challenge has been completed. So, if the player has completed all five levels, an informative and supportive pop-up message with the final score is displayed with the message that the game came to an end (Figure 45).

Figure 45: Final Message of the game

4.3.2 Timer

Once the player starts a level a timer is set to stimulate the player to solve the puzzle as quickly and effectively at the same time as possible and evaluate the player's performance from the perspective of game design. A timer in games constitutes a general game mechanics that keeps the player on the alert to perform better and produce, more specifically in the game, quickly the correct answer. Also, the timer itself gives the player the impression of being evaluated leading to a mild and unconscious urge to perform even better.

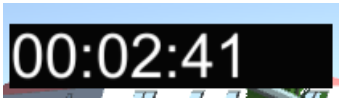

Figure 46: Timer

The timer in the game is completely developed by using a C# script attached to the main controller of each level. The time spent in a level is calculated in the script and is displayed on the screen like in Figure 46.

4.3.3 Score

The score is the main mechanics of the game that is used to evaluate the player's performance and it is based on the difficulty of each level and on the time the player needed to solve the puzzle. The score is constantly visible to the player like in Figure 47 and finally at the end as already depicted in Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε. the player can be informed of the final score of all levels.

Figure 47: Score

The main objective of the score as a strategy is to encourage the player to perform better and collect as many points as possible during the gameplay. Also, the score cannot be reduced no matter the answers the player provides or the number of failed attempts. The score and the points that the player earns while progressing in the game according to the time needed at each level are described in Table 1 and follow a gradual increase as the levels follow one another.

Level	Time (sec)	Score (points)
1	< 60	150
	< 90	100
	> 90	50
2	< 60	200
	< 90	150
	> 90	100
3	< 90	250
	< 120	200
	> 120	150
4	< 180	300
	< 300	250
	> 300	200
5	< 240	350
	< 360	300
	> 360	250

Table 1: Score points according to Levels and Time

It is obvious that there is a consistency between the minimum and maximum points at each level and as the player passes from one level to another and solves more complex puzzles has the opportunity to gain more points. So, the less time the player needs, the more the points that are accumulated in the imaginary basket of points in the game. The time in seconds as defined in Table 1 is considered as a logical amount of time that a software developer with minimum experience would need. There is no maximum time defined in the levels and obviously, the player could spend as much time as he wants in each level.

In Unity's environment to calculate the score, the main controller -the "UIWinController"-, and an empty GameObject - the "Win"- object are needed. Firstly, an empty GameObject ("Win") was created to model the desired behaviour of the system when the player finds the correct answer. The "Win" GameObject acts as the controller that defines and tweaks all parameters of the game before the player moves to the next level. To perform all these actions, a script, called "UIWinController", was created and was attached to the GameObject. This practice of a GameObject followed by a script is a common case in Unity and it is widely used for producing features and mechanics in a game. The main variables that were most necessary for the script were the "Score" variable and the "Time" variable, created as public variables in the script to be able to be exposed to the overall game. So, the assignment of the points based on the time the player needed was implemented accordingly in the script as described.

4.3.4 Help

As in many interfaces, the user may need extra help during navigation and on account of this, it is a common practice to incorporate general guidelines to which the user can have easy access. So, a "Help" button was always accessible to the user and provided useful information on the game and on the requirements that had to be fulfilled for the user's answer to be considered as correct. The "Help" button was placed below the "Score and "Time" fields to be organized spatially in a way that could be convenient for the user to overview some of the details and actions of the game (Figure 48).

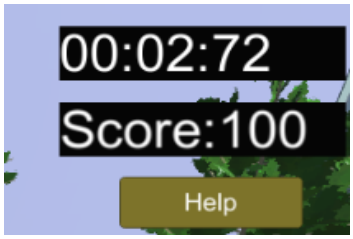

Figure 48: Help button

The information that the dialogue included was most relevant to the general game's requirements but also to the version of the interface, either drag-and-drop or typing with autocomplete (Figure 49). The user could read this information before starting the game while on the Main menu (Figure 43) and any time while playing the game. Finally, it is critical to state that the time is paused once the player decides to read the instructions and is retrieved and starts counting again once the player closes the dialogue by pressing the corresponding button.

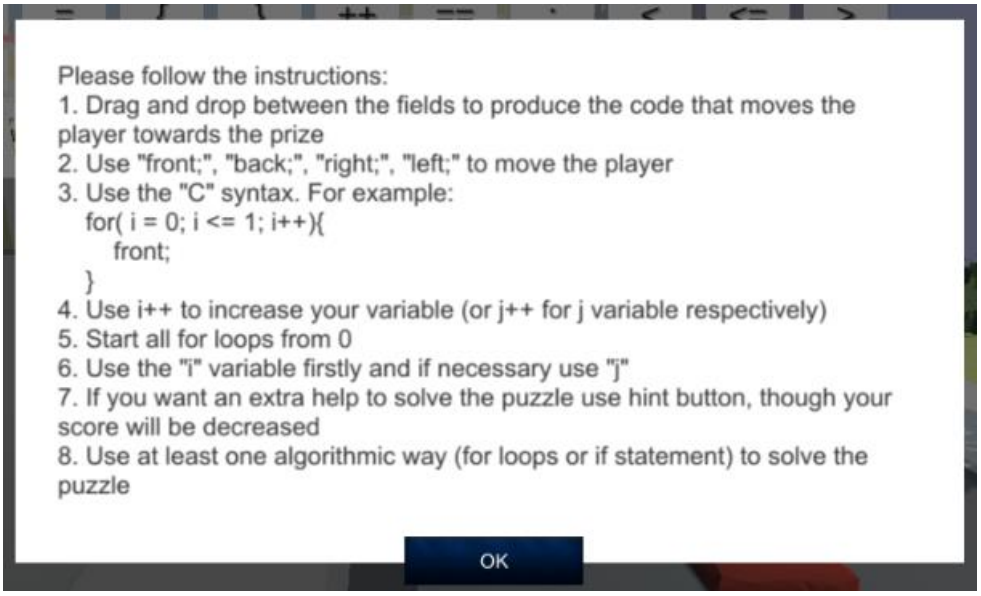

Figure 49: Dialog's information

The instructions firstly prompt the player to use the interface in a way that is intended to be used (either by dragging and dropping the fields with the commands or by typing the commands). The remaining pieces of instructions include some of the following critical requirements:

- The commands the player has access to and are feasible to perform a movement of the avatar are the commands of "front", "back", "right", "left" and should be followed by a semi-column to move the avatar towards the relevant direction.
- The syntax of the game for the programming part is the one defined by C. A random example is provided to guide the player to successfully construct the code.
- The player should use the syntax of ++ to increase the variable. So, the syntax defined by the command "i = i + 1" is not accepted in the game.

4.3.5 Hint

To help the user progress in the game and introduce gamification techniques in the implementation, a “Hint” button was added to the interface to release some guidelines of the solutions to the player. The functionality of the “Hint” button is also included in the information of the “Help” button (Figure 50) and the player is informed that since an extra help is needed to get on, a useful piece of information will be revealed and at the same time, this information will force the score obtained to be decreased. So, the extra help comes along with the effect of costing points to the player’s score. Also, all levels are supported by this extra information and this information is part of the solution to the current problem that the player struggles with. As well as the position of the “Help” button, the “Hint” button is organized to the same section below the “Hint” button.

Figure 50: Hint button

The information that is included in the Hint contains a piece of code and some useful guidelines as depicted in (Figure 51). Also, the score of the player is decreased by 50 points for all levels and since the “Hint” button has been clicked at least one time, the player’s score is not further decreased, which means that this piece of code will be always available for the whole amount of time that the player stays at the level without any further cost. Also, as well as in the “Hint” button, the time is paused when the player clicks on the button and starts counting again once the dialogue is closed.

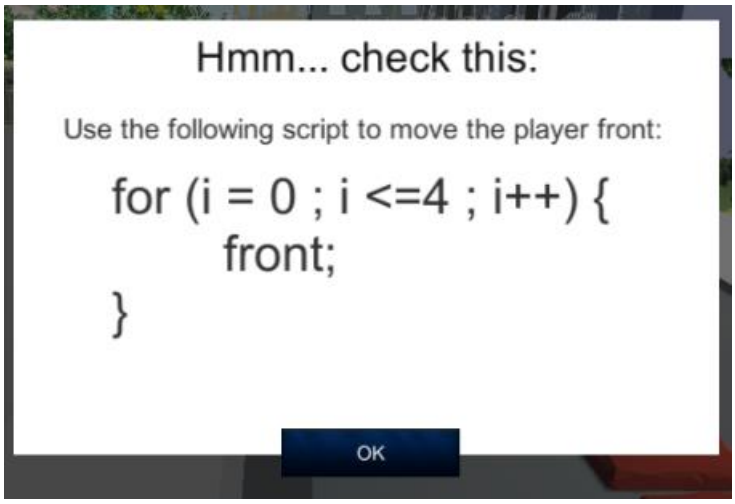

Figure 51: Hint information

Finally, the implementation and the functionality of the “Hint” button in Unity follows the same logic with which the score is calculated and assigned to the game by using an empty GameObject and a script attached to this GameObject.

4.3.6 Clear

The game's interface to provide more functionalities and prompt the user to actively play the game has incorporated a "Clear" button which can reset the user's input. This functionality is helpful for the player as to whether a wrong implementation may have been followed it would require great effort to reset the input. So, instead of manually removing all inputs, either by dragging-and-dropping or by removing the commands from the text field, this button can set the gameplay's input back to the initial state. Moreover, this functionality can be helpful for the developers as well. If something in the game does not behave as expected the user can reset the input and start producing the code again from the beginning.

The button is placed at the bottom right corner of the interface (Figure 52) next to the avatar in order not to be easily accessible or clicked by mistake. The button works for both interfaces of drag-and-drop and typing. In the drag-and-drop interface, all components are moved back to the initial canvas and in typing interface, the text field is set back to an empty character removing all characters the player may have entered.

Figure 52: Clear Button

4.4 Versions

For this study, four different versions, or mini-games, were developed and evaluated through experimental analysis. These versions were based on the same template of the game and the differences between them were the type of interaction on the user's interface and the mode that the user was playing, either by producing the code from scratch or by debugging a faulty code. Except for these differences and since these parameters were decided to be evaluated, all other mechanics like buttons, score, timer, etc. were the same. So, all players were exposed to the same challenges and the same interface as for the beforementioned mechanics.

4.4.1 Drag-and-drop / Production

During the version that combines the drag-and-drop interface and the code's production, the player was asked to drag-and-drop commands from one field to another. The first field contains all the commands that are eligible to move the avatar towards the prize leading to the completion of each level. Also, all commands remain the same from the beginning to the end of the game and do not change as the player continues playing and proceeding in the game.

Figure 53: Drag-and-drop / Production

The interface the player is exposed to is depicted in Figure 53. Also, it is important to describe the way that the commands are controlled on the user's interface each time the player decides to move a command. Firstly, whether the user decides to pick a command from the "bucket" field, the field that contains the objects with the commands, a new object is created. This decision was extremely critical for the game since in any case, the game would be very easy if the main task was to just order and line up specific commands. On the other hand, when the player decides to remove a command from the code that will be submitted and moves the command back to the first field, this command is simply destroyed back in the "bucket" field. Finally, the player can change the order of the commands and move each command that is picked up right, left, below, or before its initial position.

Figure 54: Addition of integer

To support the input of integers in the field, a button with the label "Add an integer" was created to represent this functionality (Figure 54). The buttons that were displayed since the player clicked on this button could increase or decrease the value of the integer that was created. Finally, by clicking on the "OK" button these three buttons, increase, decrease and "OK" disappear. The player submits the code by clicking on the "Execute" button and thereafter the code that the player decided on as a possible solution is evaluated. If the player

has entered a correct solution to the puzzle, then the avatar starts moving towards the prize. Otherwise, a dialogue is displayed and informs the player to check the code again.

4.4.2 Drag-and-drop / Debugging

The player is exposed to a similar interface as in drag-and-drop and production with the difference that the final field already contains a piece of code. This piece of code is faulty and cannot be the solution to the puzzle. As displayed in Figure 55 the code is similar to the final solution, though it contains syntactic or logical errors.

Figure 55: Drag-and-drop / Debugging

The player discards any superfluous commands that do not match and can drop them back in the “bucket” field. If the player thinks that the code can be re-arranged, then by starting to drag the command selected, a new empty position is created (Figure 56). This position represents the final position of the command whether it is decided to be dropped. Also, the same functionality works for the commands that contain integer values. If the player decides to alter the value of an integer, then this command must be discarded firstly and thereafter create a new one. So, a specific command with an integer cannot be selected to change its value. Another parameter of not only this variation but also of both versions of the drag-and-drop interface is that the player does not have the option to leave spaces between the commands as they are ordered sequentially, one after another.

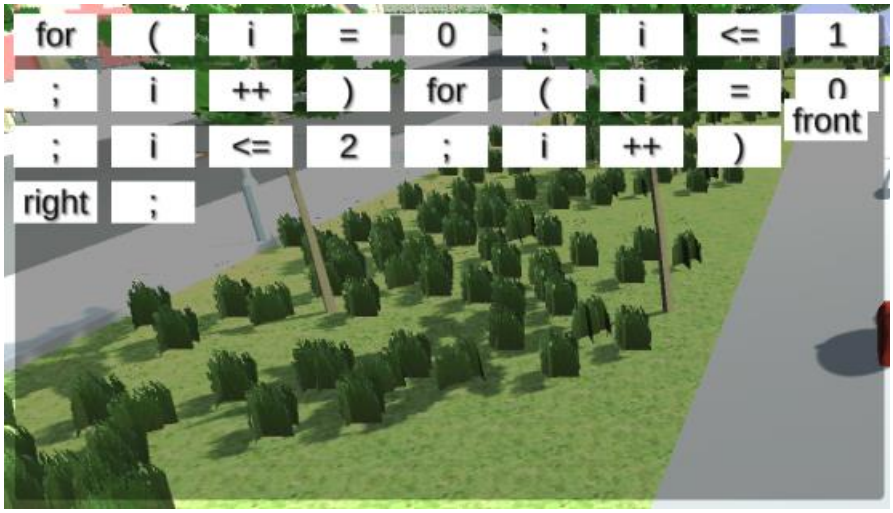

Figure 56: Rearrangement

4.4.3 Typing / Production

The player in this mini-game types the possible solution of each level by using the keyboard. So, for the player to face the challenge and solve the puzzle that is depicted on the interface, the text field should contain the code the player considers valid. To give further insights to the players on how to use the interface, there is a placeholder that informs them that they should produce the code by writing on the specific text field (Figure 57). Also, as this type of interface can be reflected as a more difficult challenge than the one on the drag-and-drop interface, the mechanics of the autocomplete method was introduced. This idea serves the game in many ways. Firstly, it gives the player the information of the possible commands and provides useful insights on how the code can be structured. Secondly, the autocomplete method is a common practice and is widely used by many code editors. It consists of an innovative way to facilitate developers and aims at their best performance while coding with minimum effort and fatigue. So, the players most probably would be already familiar with the way it works. Finally, the autocomplete, except for the effort, saves a ton of time for developers which in our case can balance a possible greater effort of having to think the command from the scratch.

Figure 57: Typing / Production

While typing the player can overview the possible command that can be used in the code on the black text field as in Figure 58. The player can complete the word by pressing the right arrow on the keyboard.

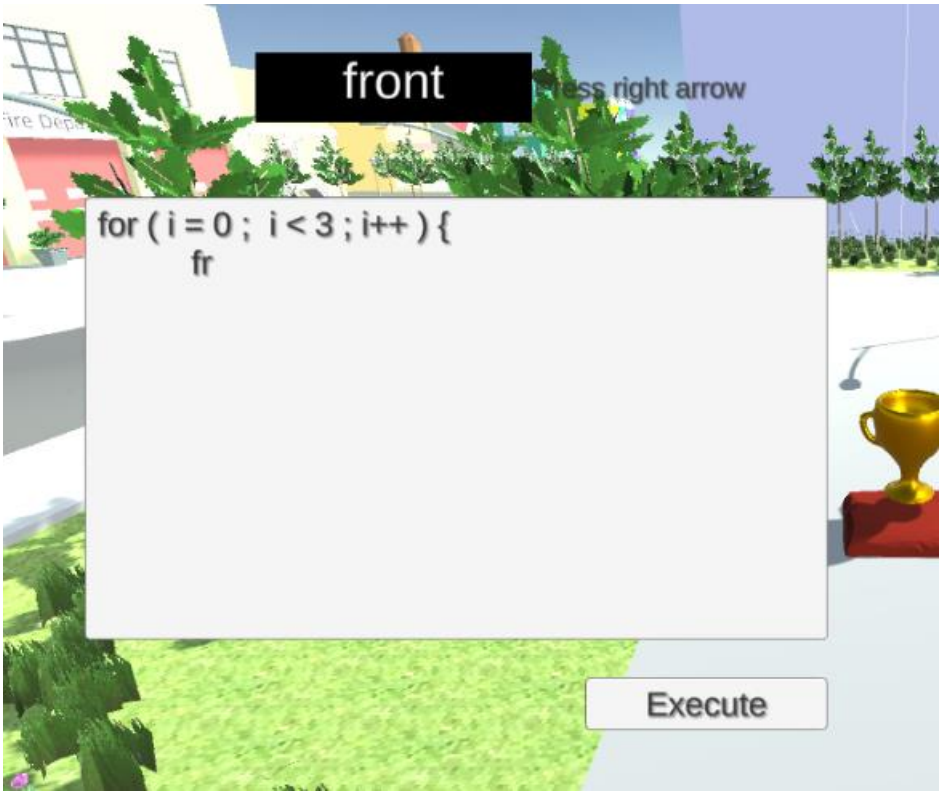

Figure 58: Autocomplete match

Thereafter, the word will be incorporated in the text and the cursor will be moved after the last character of the word as in Figure 59.

Figure 59: Word completed

4.4.4 Typing / Debugging

This interface is the same as the typing interface where the user produces the code except for the initial state of the text field of each level. As in Figure 60, the player is launched at the level and has to solve the puzzle by correcting the predefined code. Also, all mechanics of autocomplete are supported and the player has the option to correct the syntactic and logical errors or to remove the code provided and start from scratch. An important feature of this interface, and generally of the typing interface, is that the player can format the code and edit the spaces and the tabs. So, contrary to the drag-and-drop interface where white boxes are dropped between fields, the player has the freedom to create a visual representation of the code's formatting that seems convenient. The code provided in this interface was already formatted with a minimum format, though while debugging the user could change this format and use any format that seemed more convenient and with which he could decode the information of the code easily. Finally, this interface seems more realistic to most developers who are used to interacting with the keyboard to produce or debug the code.

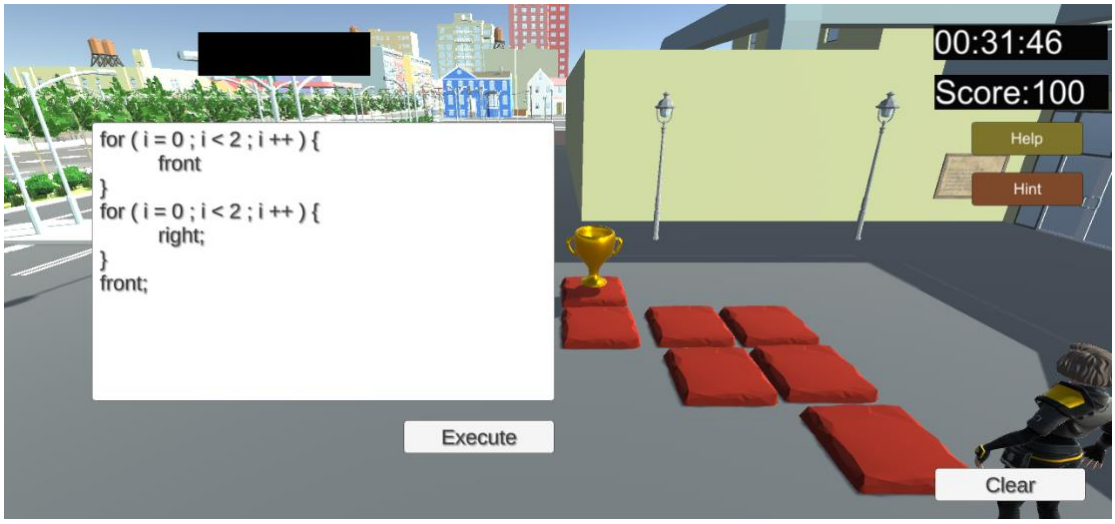

Figure 60: Typing / Debugging

4.5 Game's Challenges

All mini-games consist of 5 levels and the main task is included in each level. All tasks are slightly more difficult than the task of the previous level and require the player to move the avatar towards the prize.

4.5.1 Level 1

The player at Level 1 (Figure 61) moves the avatar towards the prize following the direction that moves the avatar front. The task was planned and estimated to be as easy as possible as it is the first challenge that the player faces in the game and except for the task and challenge of this level, we weigh in the time the player needs to become familiar with the environment, the avatar, the syntax of the language, the goal of the level, the interface, etc. So, the task of simply moving the avatar a few steps front would not discourage the player at the beginning of the game. The player writes the code that moves the avatar until the last tile, meaning that the avatar must touch the prize. The "Hint" button provides the player with the solution to move forward in the game so the progress will not be interrupted. Also, two solutions can be accepted in the game and both of them require the player to use a "for" statement. Finally, to complete successfully this level, the player should use the for loop inevitably, move the avatar exactly five steps forward and use the "front" command for this move.

Figure 61: Level 1 / Drag-and-drop production

4.5.2 Level 2

The player during the second level of the game (Figure 62) combines two different moves and directions to complete it. The player uses computational thinking for the movement towards the front direction and the movement towards the left direction afterwards by translating these movements into not only one or more for loops or if statements but also the commands of movement. Many solutions are accepted since the code does not have any syntactic errors and performs the movement of three steps forward and then two steps left.

This level was designed to be a bit more difficult than the previous one to introduce gradually the player into the game and help him understand the rules and avatar's movement. The hint at this point is suggesting moving the player front and then left by providing exactly the correct script for the second movement. In Figure 62 the code provided, in which errors are incorporated, suggests using two for loops to perform the avatar's movement. Though, the player has the freedom to provide another solution and can fix the code by suggesting another implementation. Finally, the acceptance of different solutions is a valuable feature of the game, especially as the player progresses in the game, as different players are thinking of different implementations which work better for them or they may be more fluent in using specific statements and indexes.

Figure 62: Level 2 / Drag-and-drop debugging

4.5.3 Level 3

The player faces a more complicated problem where the puzzle and the route are designed through the tiles depicted on the floor. Here, the player comes up with ideas on how to handle different directions and at the same time the iterations needed in each direction. Also, in Figure 63 the player overviews a piece of code written in the text field, some of which may be correct, and the different ways that commands can be used during the game become more prominent. This level serves more the objective of the game for the player to acquire the knowledge of how the directions can be used to move the avatar and provides the idea that single commands can be used, especially when the avatar is not required to perform multiple steps on this direction.

As well as in other levels, the player has the freedom to provide any implementation that moves the avatar to the final tile that contains the prize. If statements can be used in this level as in Level 2 to move the player conditionally either front, left, or right under a global for statement. Complementary to if statements, else if statements constitute a solution that can be accepted. When the player clicks on the “hint” button the user is prompted to move the avatar front and left firstly and afterwards the code that moves the player front again and then right is provided. Finally, a little help during the game is always necessary so that the user has a feeling of safety and confidence in the game.

Figure 63: Level 3 / Typing debugging

4.5.4 Level 4

This level does not constitute a level with great difficulty in solving the puzzle, though it is a level that contains two possible paths towards the prize so the player faces possibly the challenge to solve the puzzle and find the route that seems more convenient. Both paths can be accepted as solutions and as depicted in Figure 64 once the user is playing the version of debugging, he is provided with two “for” loops that can be tweaked along with the correct commands.

Also, the avatar does not need to walk through all tiles but only from those that a path towards the prize can be designed. The “hint” in this level prompts the user to think of the simplest path to start designing this route. Since the player can design the left possible path in the game, he is informed that only one for loop is necessary for the solution as all other commands can be represented as single commands. The for loop that is inferred in the hint is the loop that moves the player two steps forward. Though, the advice of finding the simplest path can be ambiguous as the player may think of another path as simpler than the one that was provided to help. However, there lies the difficulty of this level, that is to say, the player should be ready to abstract the solution in the game and feel confident even if the instructions provided may lead to another solution that he has already designed.

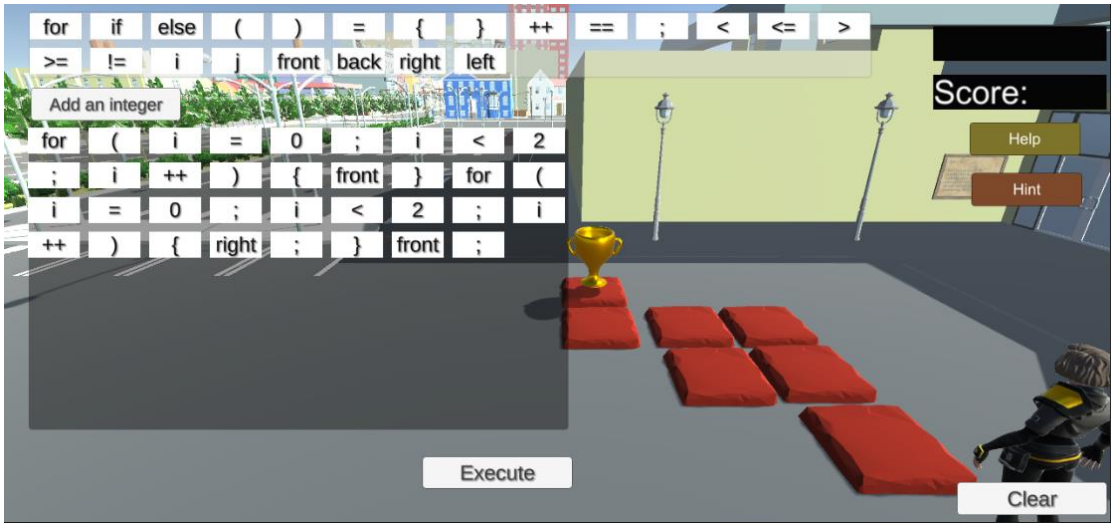

Figure 64: Level 4 / Drag-and-drop debugging

4.5.5 Level 5

This is the final level of the game and it is supposed to combine all the knowledge that the player has acquired until now. The player uses the directions of front, right, and back to reach the prize (Figure 65). It is not an extremely difficult level, though the player should find the correct indexes that must be used in the for loop and employ for the first time the direction that moves the avatar backwards.

The hint button in this game provides the simplest solution in the game and prompts the player to use a for loop for the forward movement and afterwards a block of four single commands instead of two more for loops. Though this is the final level of the game, this level's objective is to make the player feel satisfied with the overall experience and make him feel confident for completing the game successfully.

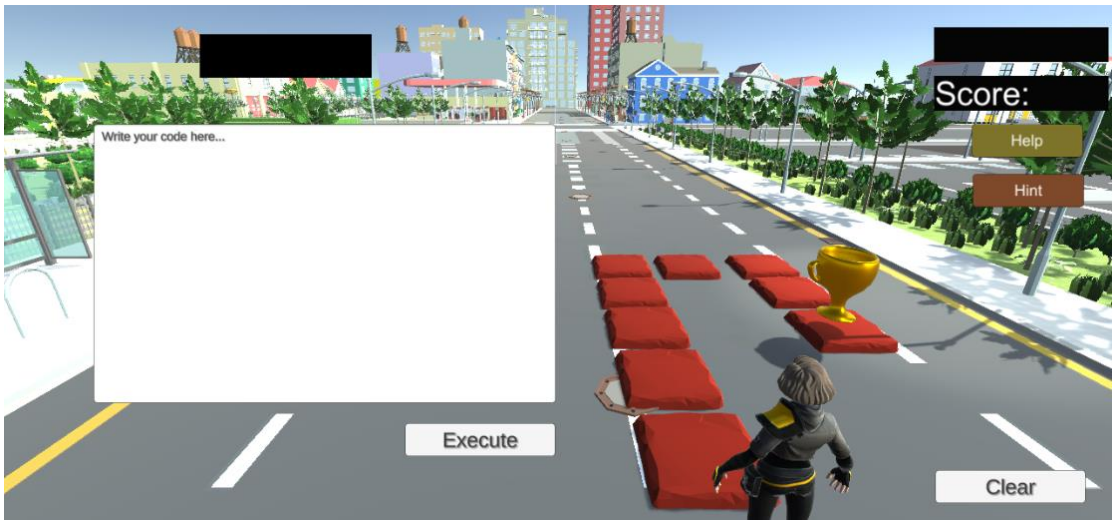

Figure 65: Level 5 / Typing production

4.6 Technical Limitations

To implement the functionalities supported in the game, we developed many components from scratch, like drag-and-drop and autocomplete in the game, as Unity does not have native libraries eligible to fit and meet the requirements set. Except for the development of these

features, another challenge was to think of and design an efficient way to evaluate user's input with the correct answer at each level.

The main question to answer for the above problem was concluded to the following question: "Should we let the player have access to the code and execute directly player's input in the game?". That means that anything the player could write would directly pass to the code as an executable part of it. This functionality is supported by the eval() function[32] which evaluates a string and executes it. Though, the eval() function introduces insecurities and makes the system vulnerable to attackers who may supply the program with strings that change the code's parameters. Also, except for the security issues, the eval() function is supported in JavaScript and not in C# which means that all implementations in Unity based on C# do not have this functionality and may migrate to JavaScript. Though, some attempts for a C# expressions parser and executor have been made and can be found in Asset Store[33].

So, keeping in mind the above limitations, C# limitation's for eval() function and overall vulnerabilities of this function, another solution for the evaluation of user's input was proposed. That solution included the incorporation of predefined options as solutions for each level. So, each time the player was executing the code, the input was compared to one of the possible solutions that were designed. As the number of solutions for each game was probably too large to be designed, this solution introduced the need for restricting player's options by setting rules in the game. Some of the rules included the limitation of starting all loops from zero and using the "i" variable firstly and if necessary the "j" variable. We decided to base the solutions on these rules, as they are already intuitive to players with even a little experience. Finally, even if a large number of possible solutions were predicted, we had in mind that some solutions may not be included in the array of possible solutions. So, when the player did not succeed to complete the level, he was informed and prompted to try another implementation whether he thinks he wrote a correct one which does not work.

To provide the players with the game, the process of building the game in Unity's program was necessary. The project was exported as a WebGL project to be eligible to run in a browser. Also, a local webserver was necessary to host the build and the extension of a Web Server for Chrome[34] was proposed. This extension serves web pages from a local network and is an open-source (MIT) HTTP server for Chrome.

4.7 Limitations

Taking into account human factors, evaluating the design and interfaces that users are exposed to is not only necessary and but also at the same time difficult and costly. Consequently, it becomes imperative to quickly test the designs and assess their differences without involving the users. GOMS (Goals, Operators, Methods, Selection Rules) is a modelling technique that designers use to model user behaviour. Briefly, Methods are used to achieve Goals. A Method is a sequential list of operators and if there is more than one Method that can be employed to achieve a Goal, a Selection Rule is invoked to determine what Method to choose. KLM (Keystroke-Level Model) is the simplest variant of GOMS and was proposed by Card, Morn and Newell [27].

[32] https://en.wikipedia.org/wiki/Eval
[33] https://assetstore.unity.com/packages/tools/visual-scripting/c-eval-56706
[34] https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb

Based on KLM we analyzed the time that the user needs to complete each level in the game taking into consideration both interfaces and activities. The results are based on the same solution for each level and the calculation of the time starts since the user positions the hand on the mouse and starts producing the code and ends once the user produces the code and clicks on the execution button. The mental operator (M), the mental act of routine thinking, is excluded since it depends mostly on the users and the game and not on the interface. The operators used for the calculation of the time needed are typing (T), pointing (P, 1.10 sec), pressing or releasing the mouse (B, 0.10 sec) and the time the user needs to move the hands between keyboard and mouse (H, 0.40 sec). For the keystroke operator, we calculate the time for an average skilled typist (0.20 sec) and all participants are younger adults under 36 years old.

We consider also that the user in the typing interface starts by typing English characters, so the case that a player starts with another language and changes to English in his keyboard's input is not included. Also, in the typing interface, the player is free to leave as many spaces as he wants or format the text in any way. For the analysis of the KLM, we use a convenient and more popular format for the programmers, by leaving spaces between the commands and the special characters and changing lines each time a bracket is used. Also, in the typing interface, we have assumed that the player uses completely the keyboard without changing the hand position from the keyboard to the mouse.

In Table 2 the time needed for all levels is depicted for both cases of the drag-and-drop interface and the typing interface. By using the analysis with the KLM model, we notice a difference between the two interfaces, with the typing interface being quicker for the user to produce the code. Though, these differences do not seem significant as they describe the actual time the player needs to complete each level without calculating the time needed to think of the solution. So, we expect a small difference between them but finally, the total time a player needs depends mainly on the mental operator (M) which is not included.

Level	Drag-and-drop (sec)	Typing (sec)
1	41.80	13.00
2	51.20	15.40
3	60.60	18.40
4	53.60	17.40
5	114.80	32.20

Table 2: KLM for production activities

In Table 3, the time the player needs to produce a specific solution by debugging the code provided is depicted. For the calculation of the time for the typing interface, we assume that the player uses the mouse to move the cursor between the positions where the errors are present. This condition is not necessary though, and we have assumed it as a reference for the calculation. As well as in the interface where the player performs the production of the code as the main activity, we conclude that also during the debugging activities, the player needs more time in the drag-and-drop interface at all levels.

Level	Drag-and-drop (sec)	Typing (sec)
1	18.10	10.80
2	59.50	20.70
3	59.30	22.20
4	59.50	15.90
5	121.80	39.10

Table 3: KLM for debugging activities

Except for the KLM model, other limitations may also affect the performance of the player in the game while investigating the effectiveness, efficiency and usability of the drag-and-drop and the typing interface. Firstly, except for the KLM model, to measure the differences between the performance of the players in the interfaces, a useful idea would be to conduct a calibration pre-test. So, each participant could perform two simple tasks, one for the drag-and-drop interface and one for the typing interfaces to calculate the differences in time between them. These tasks could be irrelevant to the game and should focus only on calculating and testing quickly the designs and evaluating at which level the total time is affected by the affordances of the interface. Also, keeping in mind that most developers use code editors and as a result the keyboard to produce the code in real-life problems, we consider that there is also the limitation that most participants would be mainly familiar with the typing interface and less with a drag-and-drop interface. This factor is especially important for participants who have acquired many years of experience in programming.

5 Experimental Analysis

The experimental analysis includes the conduction of the experiments, the data collection, the metrics used to evaluate the system, and the analysis of the experimental results. The main objective of the study is to measure mostly the effectiveness, efficiency, and usability of each system. Also, dependent and independent variables are defined and also demographics data were collected. Qualitative data were also collected including the comments of users on the interface and the overall game.

The main subject of the research is to measure the effectiveness of the serious game's design while performing tasks that include coding. So, the general goal was to define on which level the player could produce the desired output on each interface.

5.1 Design the experiment

As the field of HCI evolved, it became clear that experimental research was necessary for generating hypothesis-driven knowledge in a controlled setting. The main goal of this research is the evaluation of whether a user-interface design is better than another while the user performs certain activities that belong to the general field of software development. So, this study's goal is to compare the design of a fully mouse-controlled interface to a fully keyboard-controlled interface and evaluate the effectiveness of both on different tasks.

While comparing several user interfaces in a study, there are two main ways of assigning the participants: between-subjects and within-subjects. In the between-subjects study design, different people test each condition so that a person is exposed to only one user interface. On the other hand, in the within-subjects study design, the same person tests more than one condition in the experiment. As the learning effect was critical for the experiment, with a between-subject design, the knowledge that players may have already acquired was not an issue as they were only exposed to one condition. So, if players were exposed to two interfaces with relevant challenges, it is obvious that their reaction and performance would be better in the second experiment.

The between-subjects design also prevents fatigue effects that are caused when participants get tired. The fatigue effect was also a critical factor in the experiment. Whether participants were exposed to a fully implemented game that required concentration and mental effort, the probability of performing worse during the second experiment was high. Also, every session in a between-subjects experiment can be fairly quick. In contrast, a within-subjects experiment lasts longer as each participant is exposed to more than one design.

It is important to consider briefly the pros and cons of between-subjects versus within-subjects designs when deciding on the research methodology. However, a primary disadvantage of a between-subjects design is whenever the individuals in one group have characteristics that are different from those in another group. Also, a between-subject design requires more participants for each condition to match the statistical power of a within-subject design.

5.2 Metrics

To begin with the experimental research, it is necessary to define the predicted relationship between two variables which means actually that one thing should influence another. This fact can be seen as the effect of the independent variable on the dependent. The independent variable is the variable that can be explicitly manipulated and there is nothing a participant can do to affect it and the dependent variable on the other hand mirrors the outcome and the

data that will be gathered. The independent variable must have at least two levels. The independent variables of the experiment are described in Table 4: Independent variables.

Factor (IV)	Levels
Interface	drag-and-drop, typing
Mode	production, debugging

Table 4: Independent variables

The population of interest of this study is anyone who can understand basic concepts of computer science and computational thinking, has a prior, even elementary, knowledge and experience of software development, and can use the interfaces that are designed. The sample in the study will be thereafter based on the population that demonstrates the above qualifications.

To measure the overall usability the ISO 9241-11 was used as an approach to usability metrics. The usability metrics include effectiveness, efficiency, and satisfaction. Effectiveness, which is the main metric of the study, is defined as the accuracy and completeness with which users achieve specified goals. Efficiency is the ratio of useful work to resources expended. Finally, satisfaction addresses the comfort and acceptability of use.

5.2.1 Effectiveness

The main usability metrics for effectiveness is the completion rate and the number of errors. The completion rate can be calculated whether a participant manages or not to complete the task, for example in the specific game to reach the final Level. So, the ratio that is calculated for the study is the number of participants that completed the game, depending on the interface they used, to the total number of participants.

Another measurement involves counting the number of errors the participant makes when attempting to complete a task. These errors include unintended actions, slips, mistakes, or omissions. In the case of this study, the number of errors is defined as the total number that each participant provided to the game an incorrect answer in all five levels of the game.

5.2.2 Efficiency

Efficiency is measured in terms of task time, that is the time that the participant takes to complete a task. The time can be calculated by subtracting the start time from the end time. In this study, efficiency is measured by subtracting the time that the player is launched on the game from the time the player reaches the prize in the final level.

5.2.3 Usability

User satisfaction can be measured through standardized satisfaction questionnaires that the participant completes after the session. For the task level satisfaction, the ASQ (After Scenario Questionnaire) was used to measure how difficult the task was. The ASQ included the following questions and the answers range from 1 (strongly disagree) to 7 (strongly agree):

1. Overall, I am satisfied with the ease of completing the tasks in the game.
2. Overall, I am satisfied with the amount of time it took to complete the tasks in the game.
3. Overall, I am satisfied with the support information (help, messages, documentation) when completing the tasks.

Once the results from the questionnaire are collected, the ASQ score is calculated, which is the average of the responses in the above three questions. Higher scores are better than lower scores as the questionnaire follows the Likert scale. A lower score means that the participants found the task to be difficult or time-consuming or they felt unsupported during the game. The ASQ score is straightforward to be calculated as it includes only the task of averaging the responses received.

As the ASQ does not provide rich feedback and does not include an open-ended section, as for usability three more questions were asked:

1. What did you like most in the game?
2. What did you like least in the game?
3. What are your general comments?

The dependent variable is translated to a measured human behaviour. The study takes into consideration metrics that are associated with effectiveness, efficiency and usability. So, metrics that are considered important are summarized in Table 5.

Factor	Metric
Effectiveness	Completion Rate
Effectiveness	Number of Errors
Efficiency	Completion Time
Usability	ASQ Score

Table 5: Metrics

5.3 Limitations

Recruiting participants was a task that required time and effort and mainly because of external factors (like COVID-19 and social distancing as a consequence), there was a small number of participants. So, the final number of participants did not reach the number of participants that researchers had initially designed. Also, the experiments were conducted online, so, external factors were another parameter in the experimental environment. Ideally, the experiments were designed to be conducted in a controlled environment and not affected by factors like technical equipment, internet connection, etc. Finally, another limitation was that between-subjects experiments require a large number of participants to generate useful and reliable data so, the experimental analysis was based on a smaller sample than it was initially planned.

5.4 Experiment protocol

Since the game was fully designed and the implementation was complete, experiments that evaluate the effectiveness of the different game's versions took place. The main evaluation goal is to determine which of the two interfaces, drag-and-drop and autocomplete, is most effective not only overall but also for the certain activities of code's production and debugging that are supported. Each participant was exposed to only one of the four conditions.

All participants were members of the desired population and after recruitment was finished, it was time to think and develop the method with which participants will be assigned to the tasks. To remove any variable that may affect the experiment, participants were randomized between the different conditions of the experiment. Randomization is a critical condition of a true experiment and expresses the random assignment of treatments to the participants. Since recruiting was finished, a table was created with an index from one to the final number of participants. Next to this column of indexing the versions of the game were assigned starting from one to four and following this order sequentially. Thereafter, the sequence of

the versions was shuffled in the Excel program in a way that could not be predicted. When the experimental procedure started each participant was asked to name a number starting from one to the final index of the list. If the number referenced by the participant was not referenced by a previous participant, then the participant was assigned the version that corresponded to that index. If the index was already matched to another participant, then the participant was given the task of the next index available.

All experimental sessions were conducted online at a time that was convenient for both participants and researchers. A link to Google Meet was provided to participants to make the communication feasible and the experiments as relaxing and effortless as possible. Each participant corresponded to a separate session and no group sessions were created during the experiments. Also, they were asked not to have any other contact during the session to be as focused as possible on the experiment. All participants were kindly greeted and were introduced to the purpose of the study and the procedure. They were also informed that the screen would be recorded via the Movavi tool[35] to analyze the final results and their consent for this action and their overall participation in the study were asked. Participants were assigned to a specific condition of the experiment, following the pre-defined procedure of randomization.

Since any experiment cannot be perfect, especially under circumstances of a fully online session, technical equipment and technical problems had to be predicted and decisions to be made. So, if either the participant or the researcher lost internet connection, screen recording would not stop and both of them would try to reconnect as quickly as possible. The participant was informed about this decision and was also asked not to quit the game. If in the meantime, data were not recorded, the validity of the session would be assessed by the researcher.

All project files for each task were uploaded to a public Dropbox repository, available with the relevant URL. After the assignment of each participant to a task, a link to the project files was shared and participants were asked to download the files locally. Also, to run the project locally, a web server was necessary. The simplest and quickest solution that could easily support the local gameplay from all participants without any specific technical requirement or installation was the local installation of Google Chrome browser and Web Server for Chrome[36] extension.

All the above tasks were completed with the help of the researchers until the participant was ready to start the game. Each participant was informed explicitly at which point the game was going to start and before starting the game a brief description of the task was explained. Each task was followed by a strict and written guideline that summarized the description of their task (for example, “Complete the game by typing the commands that correct the faulty code” or “Complete the game by using the method of drag-and-drop to produce the correct code”, etc.). So, four guidelines were produced, one for each variation, and consistency between these guidelines was tried to be followed as much as possible. Also, for participants’ convenience, they were informed that during the game a “Help” button is available. The relevant guideline and the statement for the “Help” button were explicitly communicated to the participants and none of the participants received further information. During, the game if a participant asked for clarification was prompted to read the instructions within the game

[35] https://www.movavi.com/
[36] https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb

and the “Help” button, and researchers did not change the instructions or behave differently among them.

After participants completed the task, they were complimented for their contribution to the research and asked a few questions about their demographics data like gender -whether they wanted to answer-, age, experience in programming, level of studies, engagement to computer games and whether they have relevant work experience. Also, they were asked to complete the After Scenario Questionnaire (ASQ) to evaluate the usability of the interface and a semi-structured interview was also conducted about what they liked less and more in the game and provided their general comments and proposals. Finally, researchers kindly thanked each of them.

5.5 Experimental Analysis

The experimental analysis was executed in IBM SPSS Statistics 26 software[37] for demographics and tests that were necessary and executed for the analysis. The analysis in SPSS included descriptive statistics, normality tests, independent *t*-tests, hypothesis testing, chi-square tests, and reliability tests. Also, except for the statistical findings, the semi-structured interview with the participants after the session provides rich feedback on the interfaces and the overall game.

5.5.1 Demographics

For the evaluation of the design, 40 participants (33 male, 5 female and 2 prefered not to say) were recruited and except for the metrics relevant to the design, demographics data were also collected. 13 participants were under the age of 25 and 27 of them were between 26 and 35 years old. Also, 20 participants were undergraduate students, 15 graduates from bachelor studies and 5 graduates from a masters program.

Another important piece of information about participants’ characteristics was their experience in programming in years. This variable ranged between 1 and 13 years and 22.5% had 5 years of experience in programming (Figure 66). The mean value was 5.45 years(Table 6).

[37] https://www.ibm.com/analytics/spss-statistics-software

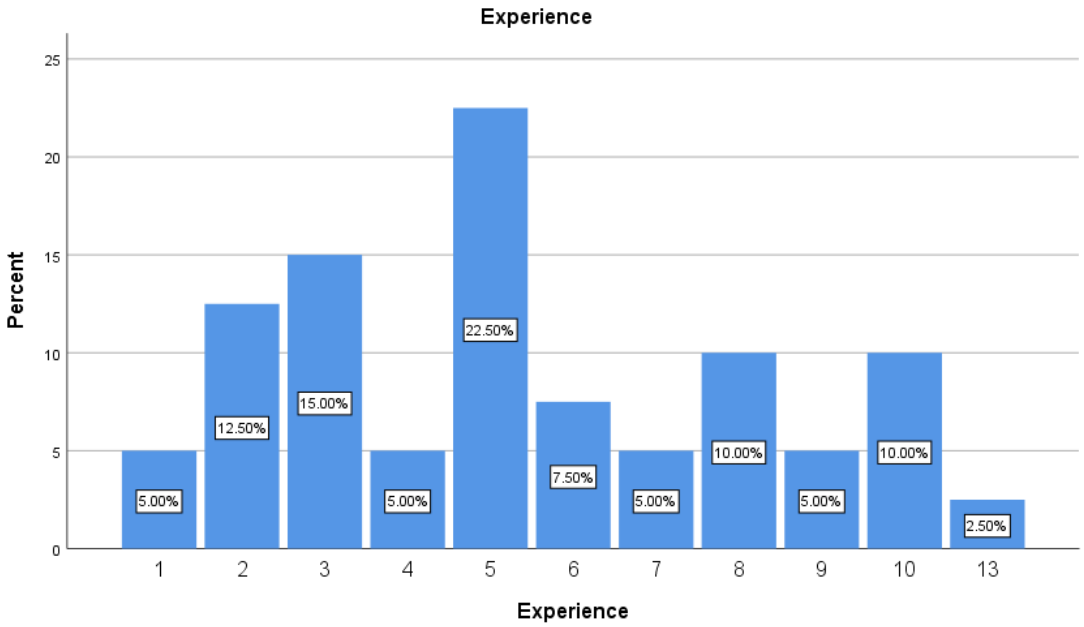

Figure 66: Participants' experience in programming (percentage)

	N	Minimum	Maximum	Mean	Std. Deviation
Experience	40	1	13	5.45	2.943
Valid N (listwise)	40				

Table 6: Participants' experience in programming (descriptive statistics)

Participants were also asked whether they have worked in an environment where programming was a task of their role. 28 of them had relevant work experience.

Regarding the time that participants spent in computer games weekly the mean value was 8.30 hours weekly (Table 7).

	N	Minimum	Maximum	Mean	Std. Deviation
Game_Play_in_Hours	40	0	50	8.30	11.469
Valid N (listwise)	40				

Table 7: Participants gameplay in hours weekly (descriptive statistics)

5.5.2 Results

The analysis is based on Hypothesis Testing, so all tests include the Null and the Alternative Hypothesis. Also, for the tests that examine the effectiveness by measuring the total errors in the game or the effectiveness by measuring the completion time, the cases that participants failed to complete the game are excluded.

5.5.2.1 Test for Normality

To start evaluating the data collected we research whether our metrics described in Table 5 are normally distributed. This condition is a requirement of many parametric statistical tests,

for example, the independent *t*-test, that data are normally distributed. We are going to focus on the Kolmogorov-Smirnov and Shapiro-Wilk tests.

The total number of errors and the completion time in minutes were checked to whether these dependent variables followed a normal distribution for the cases that participants who completed the game. So, a filter was applied in analysis excluding the cases where the participants failed to reach and complete the final level. There is no reason to think that the data of these variables differs significantly from a normal distribution and we can reject the null hypothesis that it is non-normal (p>0.05).

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Total_Errors	.113	34	.200*	.937	34	.052
Completion_Time_Min	.128	34	.169	.949	34	.118

Table 8: Normality Tests / Error_rate and Completion Time

For the normality tests of ASQ Score, we did not exclude the cases that participants failed to complete the game. There is no reason to think that the data in ASQ Score follow a normal distribution and we have failed to reject the Null Hypothesis (p<0.05). Finally, we can be confident that our data for the total number of errors in the game and the completion time is normally distributed.

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
ASQ_Average_Score	.147	40	.029	.907	40	.003

a. Lilliefors Significance Correction

Table 9: Normality Tests / ASQ Score

5.5.2.2 RQ1

Are there any differences in effectiveness related to the type of interface?

To measure the effectiveness and the statistical significance between the two types of interfaces, we examined the variable of the number of times that the participant submitted a wrong answer. All cases were examined, both for production and debugging mode, and the participants who failed to complete the game were excluded.

To answer this question we compare two independent samples, the drag-and-drop, and the typing interface. To compare these independent samples we use the Independent Samples *t*-Test. The level of significance is set to $\alpha = 0.05$.

The means of the total errors in the two interfaces are not statistically significantly different and we have failed to reject the Null Hypothesis, $t(29.01)$=0.05, $p = 0.99$.

		t-test for Equality of Means						
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference Lower	Upper
Total_Errors	Equal variances assumed	.006	32	.996	.014	2.504	-5.087	5.115
	Equal variances not assumed	.005	29.019	.996	.014	2.535	-5.171	5.199

Table 10: Independent t-Test for Error_rate

To examine the effectiveness concerning the type of interface while players create the code from scratch, we measure again the error rate during their session. The cases of debugging were excluded and sessions that players failed to complete the game were also excluded.

The means of the total errors in the two interfaces when the user creates the code are not statistically significantly different and we have failed to reject the Null Hypothesis, $t(13.25)$=-0.07, p=0.94.

		t-test for Equality of Means						
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference Lower	Upper
Total_Errors	Equal variances assumed	-.074	14	.942	-.250	3.359	-7.454	6.954
	Equal variances not assumed	-.074	13.253	.942	-.250	3.359	-7.492	6.992

Table 11: Independent t-Test for Error_rate for production mode

To examine the effectiveness in the game while players are asked to debug the code, we examine the error rate, excluding participants who were assigned the activity of creating the code.

The means of the error rate in the two interfaces when the user debugs the code are not statistically significant and we have failed to reject the Null Hypothesis, $t(10.82)$ = 0.14, p=0.88.

		t-test for Equality of Means						
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference Lower	Upper
Total_Errors	Equal variances assumed	.154	16	.879	.575	3.726	-7.325	8.475
	Equal variances not assumed	.146	10.829	.887	.575	3.944	-8.123	9.273

Table 12: Independent t-Test for Error_rate for debugging mode

5.5.2.3 RQ2

Are there any differences in efficiency related to the type of interface?

To examine the efficiency in two interfaces we measure the completion time of participants who completed the game. All cases, for both production and debugging mode, were included and grouped under the category of the interface. As in RQ1 we create the Null and Alternative Hypothesis and run a two-tailed test.

Participants in the typing interface needed overall in both tasks, production and debugging, less time (20.17 ± 10.43 minutes) compared to the drag-and-drop interface (32.58 ± 11.17 minutes), $t(32)$ =3.34, p=0.002.

	Interface	N	Mean	Std. Deviation	Std. Error Mean
Completion_Time_Min	Drag-and-drop	16	32.5813	11.17434	2.79358
	Typing	18	20.1750	10.43172	2.45878

Table 13: Group Statistics for Completion time in minutes

		t-test for Equality of Means						
							95% Confidence Interval of the Difference	
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Completion_Time_Min	Equal variances assumed	3.348	32	.002	12.40625	3.70605	4.85727	19.95523
	Equal variances not assumed	3.334	30.887	.002	12.40625	3.72152	4.81503	19.99747

Table 14: Independent t-Test for Completion_time

The means of the completion time in the two interfaces when the user creates the code are not statistically significant and we have failed to reject the Null Hypothesis, $t(13.38)$ = 1.41, p=0.18.

		t-test for Equality of Means						
							95% Confidence Interval of the Difference	
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Completion_Time_Min	Equal variances assumed	1.410	14	.180	9.03250	6.40522	-4.70533	22.77033
	Equal variances not assumed	1.410	13.383	.181	9.03250	6.40522	-4.76502	22.83002

Table 15: Independent t-Test for Completion time in minutes for production mode

To research the efficiency in two interfaces while the player debugs the code we measure again the time the player needed to complete the game. As in the above Research Questions, we create the Null and the Alternative Hypotheses.

Participants in the typing interface needed for the tasks of debugging, less time (19.69 ± 7.06 minutes) compared to the drag-and-drop interface (35.35 ± 11.00 minutes), $t(16)$=3.66, p=0.002.

	Interface	N	Mean	Std. Deviation	Std. Error Mean
Completion_Time_Min	Drag-and-drop	8	35.3500	11.00839	3.89205
	Typing	10	19.6910	7.06493	2.23413

Table 16: Group Statistics for Completion time in minutes for debugging

		t-test for Equality of Means						
							95% Confidence Interval of the Difference	
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Completion_Time_Min	Equal variances assumed	3.666	16	.002	15.65900	4.27156	6.60369	24.71431
	Equal variances not assumed	3.489	11.410	.005	15.65900	4.48769	5.82474	25.49326

Table 17: Independent t-Test for Completion time in minutes for debugging mode

5.5.2.4 RQ3

Are there any dependencies between the interface and the failure rate?

To examine whether the failure rate and the type of interface are associated with each other, we perform the chi-square test, which is appropriate for this task.

We cannot reject the null hypothesis that asserts that the two variables are independent of each other, *p*=0.376.

	Value	df	Asymptotic Significance (2-sided)	Exact Sig. (2-sided)	Exact Sig. (1-sided)
Pearson Chi-Square	.784[a]	1	.376		
Continuity Correction[b]	.196	1	.658		
Likelihood Ratio	.797	1	.372		
Fisher's Exact Test				.661	.331
Linear-by-Linear Association	.765	1	.382		
N of Valid Cases	40				

a. 2 cells (50.0%) have expected count less than 5. The minimum expected count is 3.00.

b. Computed only for a 2x2 table

Table 18: Chi-Square Test for failure rate and interface's type

5.5.2.5 RQ4

Are there any dependencies between the interface and usability?

To study the usability of the interfaces the ASQ (After Scenario Questionnaire) was created and both the questionnaire and the results were evaluated. Cronbach's alpha is a common measure of internal consistency ("reliability") and it is used when researchers have multiple Likert questions in a survey/questionnaire that use a scale and wish to determine whether the scale is reliable.

In Table 19 Cronbach's alpha is 0.855 which indicates a high level of internal consistency for this questionnaire.

Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of Items
.855	.854	3

Table 19: Cronbach's alpha in ASQ Questionnaire

Table 20 presents the value that Cronbach's alpha would be if a particular item was deleted. We conclude that the removal of any question would result in a lower Cronbach's alpha. Cronbach's alpha simply provides an overall reliability coefficient for the items of the questionnaire.

	Scale Mean if Item Deleted	Scale Variance if Item Deleted	Corrected Item-Total Correlation	Squared Multiple Correlation	Cronbach's Alpha if Item Deleted
ASQ_Satisfaction_Ease	10.55	9.023	.756	.590	.768
ASQ_Satisfaction_Time	10.55	8.818	.762	.597	.763
ASQ_Satisfaction_Support	10.15	10.079	.665	.442	.852

Table 20: Item-Total Statistics for ASQ

Finally, to compare the answers of the questionnaire between the two interfaces, we split again our data set into the answers of participants who played the drag-and-drop interface and those of participants who played the typing interface. The ASQ_Satisfaction_Ease describes the satisfaction of players with ease of completing the task. The ASQ_Satisfaction_time draws the satisfaction of players with the amount of time they needed to complete the tasks and finally, the ASQ_Satisfaction_Support describes the players' satisfaction with the support information.

In Table 21, results are depicted for the cases that participants played the game in the drag-and-drop interface. The mean of players' satisfaction concerning the ease of completing the task and the time needed are the lowest while their satisfaction in support seems to give better results.

	N	Mean	Std. Deviation
ASQ_Satisfaction_Ease	20	4.70	1.625
ASQ_Satisfaction_Time	20	4.70	1.922
ASQ_Satisfaction_Support	20	5.25	1.585
Valid N (listwise)	20		

Table 21: Statistics for Drag-and-drop interface

In Table 22, the same results are depicted for the typing interface. Overall, in all three questions, the typing interface seems to perform better in terms of usability. Finally, since the support information in both interfaces was the same, players' satisfaction for this parameter does not differ extremely.

	N	Mean	Std. Deviation
ASQ_Satisfaction_Ease	20	5.45	1.701
ASQ_Satisfaction_Time	20	5.45	1.432
ASQ_Satisfaction_Support	20	5.70	1.658
Valid N (listwise)	20		

Table 22: Statistics for Typing interface

5.5.3 Interview Findings

At the end of the experiment, each participant was asked a few questions, except for the demographic data. These questions included what they liked more and less in the game, and describe their general comments about the game. The questions were open-ended questions and participants could answer and report more than one element in each question.

In Table 23 the answers of what users liked most are reported and answers of all participants are included (N=40). 17 participants reported that they liked the graphics in the game, including the representation of the avatar and the city environment. 14 participants also reported that they liked the content of the game, meaning the goal of the game to write the actions by using programming and algorithmic logic and the way that the player and the prize were connected via coding. 20% of the participants mentioned the learning outcome of the game, stating that it can be a good and pleasant introduction to programming. 7 participants stated that the game included a smooth and easy interface and that they had a pleasant user experience. 6 participants mentioned the complexity and the difficulty of the game describing it as challenging to solve the puzzle and find the right paths they had to code even it seemed easy. Finally, some participants also liked the interface, the environment, the animations the feedback that the game provides and the mechanics that included mostly the score, help and the hint functionality.

Type	N	Percentage (%)
Graphics	17	42.50
Content	14	35.00
Learning Outcome	8	20.00
Usability	7	17.50
Complexity/Difficulty	6	15.00
Interface	5	12.50
Environment	3	7.50
Interactions	2	5.00
Animation	1	2.50
Feedback	1	2.50
Mechanics	1	2.50

Table 23: Answers of what users liked most

However, complementary findings arose from the question of what participants liked less in the game. As the answers differentiate, we decided to split these answers into what participants like less in general and what participants like less concerning the way that they interacted with the code.

Results of the answers in the first group are depicted in Table 24. 9 participants out of 40 reported that they experienced problems with the orientation or direction and what commands they should use each time to move or turn the player. 5 participants wanted more advanced debugging actions and would like the code they wrote to be automatically executed without being complete or accurate. Also, 2 participants found the feedback insufficient as they would like more information on the errors each time they entered an incorrect answer. Finally, some participants did not find the interface responsive enough, did not like the rules of the game or needed more information in the game while playing.

Type	N	Percentage (%)
Orientation	9	22.50
Debugging	5	12.50
Feedback	2	5.00
Responsiveness	2	5.00
Rules	1	2.50
Information	1	2.50

Table 24: Answers of what users liked less (generally)

To analyze the results concerning the interface, we split the participants into the group of the drag-and-drop interface (N=20) and the typing interface (N=20). Code's creation in Table 25 represents the way that users could provide the code as input in the game, either by typing or by dragging and dropping the commands. Code format describes the need of the participants to format the code, add more lines, add breakpoints in the lines, spaces, etc. So, from Table 25 it is clear that most participants in the drag-and-drop interface did not like the drag-and-drop interaction with the code and also a great percentage of them needed to format the code. None of the participants in the typing interface mentioned the above issues, In the typing interface most participants noticed the autocomplete functionality, but they did not use it. Though, it seemed that this functionality served more as guidance on the commands that they could use.

Type	N (total)	N	Percentage / drag-and-drop (%)
code's creation	20	11	55.00
format	20	7	35.00

Table 25: Answers of what users liked less in the drag-and-drop interface (code)

Participants were asked to express their comments and their proposals for improvement. As for the interfaces, participants who played the drag-and-drop interface provided rich feedback on the functionalities of the interface. Some of them reported that instead of the drag-and-drop would like an interface where the player clicks on the command instead of dragging and dropping. Though, this interface has limitations as it does not provide the user with the exact position that the command will be placed. Another suggestion included an interface where the user right-clicks on the position he wants to add a command and via a list that appears selects the right command. Also, some participants expressed their desire for an editor instead of drag-and-drop while others noted the limitations of the interface, like lack of copy and paste functionality, or selecting some commands and clearing only them.

Finally, in general, several suggestions included the incorporation of a more detailed description of the game's objectives and goals inside the game, maybe in the main menu. Also, a first-person implementation was proposed as well as the ability of the game to execute whatever the player writes irrespectively to whether it constitutes the solution.

6 Discussion and Future Work

In this chapter, the main conclusion of this study will be discussed. Also, limitations and ideas for future work are discussed as well.

6.1 Main Conclusions

HCI has started extending its methodologies to the field of Serious Games to help researchers understand better the interactions between the players and the technology and analyzes factors like not only the game's effectiveness, efficiency and usability but also the player's satisfaction. At the same time, programming and software development have been established as essential skills nowadays and more and more students or adults develop and extend their skills in computational thinking, programming and software development.

This study reviewed the existing serious games designs for software developers and to better understand the interactions of players in the game, four mini-games were also developed. The games examined the effectiveness, efficiency and usability of different game designs while

the users performed different tasks that are included in the software development process. The designs proposed were evaluated with an experimental analysis and a total number of 40 users with prior knowledge of software development participated.

Since this dissertation explores different types of games and designs, the mini-games developed were split based on two parameters: the type of interaction and the player's tasks. Following this idea, the versions that were developed are the following: a drag-and-drop interface with the tasks that required the player to produce the code from scratch, a drag-and-drop interface where the player fixes the code, a typing interface with autocomplete functionality where the player produces the code and finally a typing interface with debugging tasks. The research questions of this study focus mainly on the effectiveness, efficiency and usability of these interfaces compared to one another and under different player's tasks like production or debugging of the code. All games were puzzle games and developed in Unity. Each game consists of five levels and includes the same environment, player, mechanics and challenges.

As far as it concerns the efficiency of the drag-and-drop and typing interface, this study proved that a typing interface in a serious game for software developers is more efficient than a drag-and-drop interface. To measure the efficiency, the time that players needed to complete the game was calculated and based on the statistical analysis we found that participants in both tasks, production and debugging of code, needed less time compared to the drag-and-drop interface. Also, except for the overall analysis for both tasks, in a game where players face the challenge of debugging the code, the typing interface has proved more efficient than the drag-and-drop interface. The above findings lead to the results that players need less time to complete a game with coding challenges while they type the commands instead of dragging and dropping them. Another finding describes the fact that when players face coding challenges with debugging tasks, they have difficulty in solving the puzzles in the drag-and-drop interface compared to the typing interface.

As for the question of which interface seems to be more usable, the After Scenario Questionnaire (ASQ) was used to evaluate this perspective. It seems that in the typing interface players were more satisfied with the ease of completing the tasks compared to the satisfaction of players in the drag-and-drop interface. Finally, the interview findings also indicate that a significant percentage of participants liked the graphics in the game and some of them mentioned its learning outcome and importance. As far as the interface and the way the participants produced the code, players in the drag-and-drop interface did not like the interaction with the code and preferred not to drag-and-drop the commands. Also, it seems that the code format, mentioned by a great number of participants, is necessary as they needed to add breakpoints or split the code into separate lines.

6.2 Limitations

The most important limitation of all was the small number of participants which restricted the statistical analysis. Though, the interviews that followed provided rich feedback for the games and valuable insights on how the interfaces and the overall game could be enhanced. Also, another limitation of the experiment is caused because of the target group of the participants. Participants needed to have elementary knowledge of programming to join the experiment. Finally, there was a lack of relevant research on the field as there are not many publications and experiments that examine the differences between the interfaces in serious games for software developers. This factor made the research of this thesis more difficult as there were

no guidelines that could help the set-up of the experiment and the interpretation of the results.

6.3 Future Work

This thesis can be a useful guide for the creation of serious games for software developers with variant backgrounds. A future idea can be based on this thesis and extend the game that is already designed. For example, more functionalities and actions can be included in the game except for the movement of the player. The game can be extended to include challenges where the player uses even more commands to surpass obstacles, jump above objects, etc. We estimate that the game could be even more exciting and engaging for the players by creating more challenging levels. Also, as we have already described the game uses predefined options to evaluate the solution of each level. Another solution is to expand this dictionary of solutions and provide more possible solutions to the user. Also, the categorization of the solutions could be supported so that more advanced and difficult solutions to think would reward more the user than simpler solutions.

Another idea is to expand the game and incorporate it as additional educational content in a computer science course in a university or upper classes of high school. This requires, though, further implementation of the game and the addition of more levels and functionalities.

A final proposal could be based on further implementation or replacement of the drag-and-drop interface and the incorporation of the ideas proposed by participants. This could be an interface where the user can click instead of drag-and-drop since it has been an idea that was proposed during the experiments.

References

[1] S. Mylly, M. Rajanen, and N. Iivari, "The Quest for Usable Usability Heuristics for Game Developers," in *Advances in Information Systems Development*, vol. 39, A. Siarheyeva, C. Barry, M. Lang, H. Linger, and C. Schneider, Eds. Cham: Springer International Publishing, 2020, pp. 164–181. doi: 10.1007/978-3-030-49644-9_10.

[2] J. M. Wing, "Computational thinking and thinking about computing," *Phil. Trans. R. Soc. A.*, vol. 366, no. 1881, pp. 3717–3725, Oct. 2008, doi: 10.1098/rsta.2008.0118.

[3] D. I. Cordova and M. R. Lepper, "Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice.," *Journal of Educational Psychology*, vol. 88, no. 4, pp. 715–730, 1996, doi: 10.1037/0022-0663.88.4.715.

[4] K. E. Ricci, E. Salas, and J. A. Cannon-Bowers, "Do Computer-Based Games Facilitate Knowledge Acquisition and Retention?," *Military Psychology*, vol. 8, no. 4, pp. 295–307, Dec. 1996, doi: 10.1207/s15327876mp0804_3.

[5] R. Garris, R. Ahlers, and J. E. Driskell, "Games, Motivation, and Learning: A Research and Practice Model," *Simulation & Gaming*, vol. 33, no. 4, pp. 441–467, Dec. 2002, doi: 10.1177/1046878102238607.

[6] D. Church, *Formal Abstract Design Tools*, Gamasutra, Jul. 1999. Accessed on: Nov. 15, 2020. [Online]. Available: https://www.gamasutra.com/view/feature/131764/formal_abstract_design_tools.php

[7] C. Johnson *et al.*, "Game Development for Computer Science Education," in *Proceedings of the 2016 ITiCSE Working Group Reports*, Arequipa Peru, Jul. 2016, pp. 23–44, doi: 10.1145/3024906.3024908.

[8] A. Leigh, *GDC 2012: Sid Meier on how to see games as sets of interesting decisions*, Gamasutra, Mar. 2012. Accessed on: Nov. 18, 2020. [Online]. Available: https://www.gamasutra.com/view/news/164869/GDC_2012_Sid_Meier_on_how_to_see_games_as_sets_of_interesting_decisions.php

[9] Malone, Thomas. (1981). What Makes Things Fun to Learn? A Study of Intrinsically Motivating Computer Games. Pipeline. 6.

[10]Arawjo, Ian & Wang, Cheng Yao & Myers, Andrew & Andersen, Erik & Guimbretiere, Francois. (2017). Teaching Programming with Gamified Semantics. 4911-4923. 10.1145/3025453.3025711.

[11]Anderson, John & Corbett, Albert & Koedinger, Kenneth & Pelletier, Ray. (1995). Cognitive Tutors: Lessons Learned. Journal of the Learning Sciences. 4. 167-207. 10.1207/s15327809jls0402_2.

[12]E. Soloway and James C. Spohrer. 1988. Studying the Novice Programmer. L. Erlbaum Associates Inc., USA.

[13]Booth, S. (1992). Learning to program : a phenomenographic perspective.

[14]Rugelj, Jože & Zapušek, Matej. (2013). Learning programming with serious games. Transactions on Game Based Learning. 10.4108/trans.gbl.01-06.2013.e6.

[15] Computational Thinking in Educational Activities

[16]Kahn, K. (1996). ToonTalk - An Animated Programming Environment for Children. *J. Vis. Lang. Comput., 7*, 197-217.

[17]M. J. Lee *et al.*, "Principles of a debugging-first puzzle game for computing education," *2014 IEEE Symposium on Visual Languages and Human-Centric*

Computing (VL/HCC), Melbourne, VIC, 2014, pp. 57-64, doi: 10.1109/VLHCC.2014.6883023.

[18]T. Mitamura, Y. Suzuki and T. Oohori, "Serious games for learning programming languages," *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Seoul, 2012, pp. 1812-1817, doi: 10.1109/ICSMC.2012.6378001.

[19]Plotkin, G. (2004). A structural approach to operational semantics. *J. Log. Algebraic Methods Program., 60-61*, 17-139.

[20]Ashcraft, Mark. (2002). Math Anxiety: Personal, Educational, and Cognitive Consequences. Current Directions in Psychological Science - CURR DIRECTIONS PSYCHOL SCI. 11. 181-185. 10.1111/1467-8721.00196..

[21] Sam, Hong & Othman, Abang & Nordin, Zaimuariffudin. (2005). Computer Self-Efficacy, Computer Anxiety, and Attitudes toward the Internet: A Study among Undergraduates in Unimas.. Educational Technology & Society. 8. 205-219

[22] J. S. Atherton. Learning and Teaching; Bloom's Taxonomy. Online: www.learningandteaching.info/ learning/bloomtax.htm, 2011

[23] Fuller, Ursula & Johnson, Colin & Ahoniemi, Tuukka & Cukierman, Diana & Hernán-Losada, Isidoro & Jacková, Jana & Lahtinen, Essi & Lewis, Tracy & Thompson, Donna & Riedesel, Charles & Thompson, Errol. (2007). Developing a computer science-specific learning taxonomy. ACM SIGCSE Bulletin. 39. 152-170. 10.1145/1345375.1345438.

[24]Anderson, J. R. (1982). Acquisition of cognitive skill. Psychological Review, 89, 369-406..

[25]Kraiger, Kurt & Ford, J.. (1993). Application of Cognitive, Skill-Based, and Affective Theories of Learning Outcomes to New Methods of Training Evaluation. Journal of Applied Psychology. 78. 311-328. 10.1037//0021-9010.78.2.311.

[26] Bellotti, Francesco & Berta, Riccardo & De Gloria, Alessandro. (2010). Designing Effective Serious Games: Opportunities and Challenges for Research. International Journal of Emerging Technologies in Learning. 5. 10.3991/ijet.v5s3.1500.

[27]S. K. Card, T. P. Moran, and A. Newell, "The keystroke-level model for user performance time with interactive systems," *Commun. ACM*, vol. 23, no. 7, pp. 396–410, Jul. 1980, doi: 10.1145/358886.358895.