International Journal Of Engineering Research And Development
e- ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 20, Issue 11 (November, 2024), PP 608-618

Development of a Serious Game to Assist in Teaching Programming in Introductory Courses

Rafael Aparecido Marinho Capodeferro, Carlos Eduardo do Amaral, Lucas de Morais Pereira, Flávio Cezar Amate*
Department of Informatic – Instituto Federal of São Paulo - IFSP
**Corresponding Author*

ABSTRACT
The increasing difficulty students face when learning programming logic has led to high rates of retention and dropout in introductory technology courses. To address this challenge, we developed a serious game designed to facilitate the learning of programming logic through an engaging, immersive experience. The game incorporates a non-linear learning approach, allowing students to progress at their own pace and revisit specific topics as needed. This flexibility supports students who may struggle with linear course structures, helping them to reinforce key concepts without feeling constrained. Additionally, the game employs a training and reuse model, enabling learners to continuously practice programming logic by returning to the game for further exercises. With cross-platform compatibility, the game can be compiled and deployed on various operating systems, making it accessible to a broad range of students. Preliminary results suggest that the game effectively aids students in mastering fundamental programming logic, potentially reducing dropout rates in technology courses and strengthening foundational skills.
Key-words: serious games, programming logic, immersive learning, non-linear learning, dropout prevention.

Date of Submission: 09-11-2024 Date of Acceptance: 21-11-2024

I. INTRODUCTION

When studying programming logic, students not only practice logic related to algorithms but also develop skills in other activities. Often, solving a problem requires concepts from other fields of knowledge. Therefore, it is essential to have a solid foundation in programming logic, as it enables a broader perspective on potential problem-solving approaches.

Although programming logic is a crucial concept, many students in schools and universities face significant challenges when first encountering the subject, leading to high rates of retention and dropout. According to Souza and França (2013), technology courses are in high demand; however, the complexity of concepts taught in these courses contributes to increased dropout rates. Additionally, students may reach a certain point in their studies without the necessary knowledge in programming, which complicates their understanding of other subjects, (BAIST & PAMUNGKAS, 2017).

Therefore, the creation of a tool to support the study of programming logic is necessary. To this end, an educational game will be developed. According to Moratori (2003), games can capture students' attention, making them an ideal teaching tool. By employing various strategies, such as gamification and a sense of reward, it is possible to engage the player and reinforce essential programming logic concepts.To Salomão et al. (2017), the serious games can motivate students and enhance educational outcomes.

II. MATERIAL AND METHODS

Programming Logic and Algorithms

The technology market seeks professionals capable of solving multidisciplinary problems. In a company or project, a programmer may need to develop a platform for issuing invoices, and later in their career, the developer might participate in creating a platform for submitting academic articles. Each problem, therefore, has its unique characteristics.

According to Cormen et al. (2022), practical applications of algorithms are everywhere. For instance, they enable the development of tools capable of analyzing massive datasets with billions of chemical bases that make up human DNA. Another example provided by the authors is in online commerce, or e-commerce, which allows the buying and selling of products remotely, worldwide.

Programming logic is related to the abstract and logical thinking required to design algorithmic structures and computational problem-solving solutions. In other words, programming logic is the reasoning a developer uses to solve a problem. Therefore, regardless of the programming language being used, programming logic can be applied as it operates at an abstract level of thought.

The algorithm is the concrete representation of programming logic. When a problem's solution remains abstract in the programmer's mind, the computer cannot interpret it. It is therefore necessary to translate logical thinking into a sequence of instructions called algorithms, making that thought process accessible to the computer.

Retention and Dropout in Technology Courses

Retention occurs when a student remains in a given course period longer than expected. This can happen for various reasons, including academic difficulties, financial issues, family problems, lack of motivation, among others. Additionally, retention affects students' self-esteem regarding their learning progress and may increase the likelihood of dropout.

Dropout is when a student leaves their studies before completing the cycle or period of elementary, secondary, or higher education. Dropout can occur due to similar factors as retention. When a student stops studying, it impacts their social relationships in the educational environment as well as their knowledge development.

Programming logic and algorithms present a challenge for students in fields that require these skills. One of the reasons behind this difficulty is the amount of prerequisite knowledge necessary to understand the subject, as well as the learning of tools that form the developer's work environment, known as Integrated Development Environments (IDEs) (Rocha et al., 2013).

According to Junior et al. (2020), retention and dropout rates are high in introductory courses, which is a concern for educators in the field. Research conducted by Filho et al. (2018) showed that, from 2015 to 2017, the failure rate was 52% in the Information Technology course at UFJF, a course that includes algorithms and programming logic.

Serious Games

Serious games are digital games that go beyond entertainment. They can assist in various fields, such as serious games applied in healthcare, like Depression Quest, created by Zoe Quinn, which aims to help people suffering from depression. They can also be used in education, such as Meister Cody, a game that helps children learn mathematics (Caserman et al., 2020).

Today, most people in Brazil use technology. According to data from a study conducted by IBGE (Brazilian Institute of Geography and Statistics) in 2021, around 90% of households have internet access, with cell phones being the most used device. Additionally, the study shows that among people aged 10 and older, 90.3% are students who access the internet. This data indicates that digital media are increasingly accessible and frequently used by the population.

Therefore, the application of digital serious games to a population that is increasingly using technology is an important aspect. According to Yusoff et al. (2009), digital educational games are effective teaching methods, as the current generation of students grows up and develops in a digital environment, which facilitates comprehension and normalizes the use of technology.

Game Engines

Game engines are platforms that facilitate the completion of tasks related to game development, such as rendering, physics for environments and characters, animations, collision objects, and more. To achieve this, the engine provides a wide range of libraries and tools that professionals can use to bring ideas from paper to the real world (Paul; Goon; Bhattacharya, 2012).

Although the term "game engine" originated in 1990, the first game engine was only created seven years later by the Japanese group ASCII. Named RPG Maker, it was initially programmed in JavaScript and was used exclusively for creating 2D games, offering several resources, such as characters and other pre-made assets for developers (Scherer; Batista; Mendes, 2020).

Over the years, with significant technological advancements, many other engines have been developed and updated, providing even more tools to make games smoother and more realistic. For example, the new Unreal Engine 5, released in 2021, introduced improvements in rendering, animation, and simulation—features that the original Unreal Engine from 1991 could not yet offer.

Game Design

The Game Design Document (GDD) is a document created by the game development team during the production process. It contains all the necessary information to guide the team throughout the game's development, from basic information, such as characters, dialogues, and environments, to more advanced details, including level prototypes, sounds, and the technologies that will be used in the project (Motta; Junior, 2013).

Currently, three types of GDDs are commonly used in the game industry, differentiated by their level of detail and the number of pages. First, there is the single-page GDD, which provides an overview of the game and includes some visual elements. Then, there is the 10-page GDD; with more pages, it allows for greater detail on various elements of the game, such as characters, level design, summaries, and more. Finally, there is the bible GDD, which can exceed 15 pages. This version offers a complete breakdown of every aspect the game will include.

Thus, creating a GDD is an effective strategy in game production, as it helps developers stay aligned with the ideas initially proposed by the team. Regardless of its size, the primary purpose of a GDD is to ensure that all team members are consistently aligned with the project's overall vision.

RPG

Considered the first Role Playing Game (RPG) in history, Dungeons and Dragons, released in 1974 (Filho; Albuquerque, 2018), established many of the rules for traditional tabletop RPGs, as well as for computer RPGs (CRPGs). In the early days of CRPGs, traditional RPG rules were combined with the rules of other computer games. However, the primary goal of these games remained the same: overcoming challenges, whether they be battles against monsters or solving puzzles, along with character interpretation (Barton; Stacks, 2019).

With the expansion of the gaming market, RPGs have evolved to include different options. One example is a traditional RPG, such as Baldur's Gate III by Larian Studios, in which the player embarks on an adventure, making choices and interpreting the character.

Alternatively, a game may incorporate RPG mechanics, as in Horizon Zero Dawn by Guerrilla Games, where the player controls a character with limited choices but features a leveling and character progression system. Both examples fall under the RPG genre.

Development of the Game

The primary technology used was RPG Maker MV, a game engine developed by Gotcha Gotcha Games to facilitate the creation of electronic games, focusing on the Role-Playing Games (RPG) genre. This engine offers various features that aid in game development, such as character creation, map blocking, database management, controls, etc. However, despite the numerous resources provided by the game engine, it was necessary to use additional tools to assist in creating the game. In the first phase, a Game Design Document (GDD) was prepared. This document contains everything involved in the game's creation, from controls to the complete storyline.

Furthermore, the study's focus includes fundamental concepts of programming logic, such as input, processing, and output; variable manipulation; the operation of conditional structures (if, else); repetition structures (while, for); logical problem-solving; and function study.

Therefore, beyond the GDD and the game engine, it was essential to design programming logic exercises with the appropriate difficulty level. If too challenging, it could discourage the player, while if too simple, the game might become monotonous, leaving the player with a sense of not learning, which would be highly ineffective for an educational game.

Thus, the primary requirement of the system is to enable the player to learn programming logic by solving logical exercises and applying fundamental programming concepts. Additionally, as a game, it must be visually engaging, in other words, it should captivate the user's attention and provide enjoyment alongside learning.

Storyline

To strengthen the game's narrative and create an immersive setting, a script was developed incorporating a story with characters and a historical context to provide students with the challenges of the didactic objective.

The protagonist, named Ada, returns from a tiring day of classes and reviews some lectures on programming. Meanwhile, as she heads to the kitchen, something unexpected happens—a cabinet falls on her head, and she passes out. Following this series of events, a portal opens, transporting her to another dimension.

When she wakes up, Ada finds herself in a medieval dimension and encounters a wizard named Alan. He explains that several adventurers have appeared in the same way she did, but they left behind some notes to

help future travelers. Upon arriving at the guild with the wizard, Ada starts reading the notes and discovers something astonishing: the magic in the medieval dimension is similar to the programming logic she had been studying in the real world.

Ada gradually understands what is happening and asks the wizard how she can return home. He tells her it would be a mission: to study magic (programming logic) to reach the final challenge and reopen the portal. The wizard suggests that Ada help the townsfolk to practice magic—in the city, the forest, and the cave—so that she can eventually access the portal and return to her home.

III. RESULTS

Game Modeling

Initially, the modeling of functional and non-functional requirements was conducted. Functional requirements are directly related to the software's behaviors, that is, what the software must do. Non-functional requirements, on the other hand, pertain to the attributes and characteristics the software needs to have, such as security, reliability, performance, among others (WIEGERS; BEATTY, 2013), (SALOMÃO; SANTOS; GIANCOLI; AMATE, 2017). Tables 1 and 2 present the modeled functional and non-functional requirements.

Table 1 –Functional Requirements

#	**Title**	**Description**	**Status**
FR001	Character Movement	The game should allow the user to move the character byclicking with the mouse cursor.	Mandatory
FR002	AlternativeCharacter Movement	The game should allow the user to move the character using the following keys: "w" to move forward, "a" to move left, "d" to move right, and "s" to move backward	Important
FR003	Interact with NPC	The game should allow the user to interact with NPCs by approaching them and clicking on them with the mouse cursor	Mandatory
FR004	Specify Controls	The game should display the basic controls to the user, showing how to move, interact with NPCs, and navigate the map	Important
FR005	Display Dialogues	The game should display dialogues to the user when they interact with an NPC. These dialogues will present the questions or tasks the player needs to solve	Mandatory
FR006	Respond to Dialogues	The game must allow the player to respond to the dialogues presented by NPCs	Mandatory
FR007	Correct or Incorrect Response	The game should display to the player whether the response they selected in the dialogue with the NPC is correct or incorrect	Mandatory
FR008	Scoring System	The game should add points to the player's total score if their answer to a question is correct	Important
FR009	Map Transition	The game should allow the user to switch between game maps if they have the required score	Important
FR010	Display Required Score	The game should display the score required for the player to progress to the next map	Important
FR011	Block Progression	The game should not allow the user to advance to the next map if they do not have the required number of points.	Important
FR012	Map Topics	The game should feature maps covering different programming logic topics. The first map should teach the player about concepts such as variables, input, processing, output, and variables. The second map should cover conditional structures. The third map should focus on repetition structures	Mandatory
FR013	Map-Specific Questions	The game should include questions relevant to the topic each map covers, allowing the user to study and practice that specific concept	Mandatory

Table 2 – Non-Functional Requirements

#	**Title**	**Description**	**Status**
NFR001	Accessibility	The game should function on most computers regardless of the hardware quality	Mandatory
NFR002	Languages	Initially, the game should only support Brazilian Portuguese.	Important
NFR003	Platforms	The game must be compatible with both computers and mobile devices	Mandatory
NFR004	Easy Explanation	The game must explain programming logic concepts in a simple and accessible way	Mandatory

The game follows a single-player campaign model, where the unfolding of the story events serves as an additional motivator for the player to continue. Although the focus is on learning, the lighthearted narrative tone helps make the player's journey less tedious. During the game, a space at the bottom of the screen displays the dialogues and interactions the user must engage in with the NPCs. The Figure 1 shows an example of a dialogue in the game.

Figure 1 – Dialogue Mechanic in Game Interface

A more natural language style has been implemented, adding simplicity and a touch of humor to the dialogues. The goal is to make the concepts easier to understand, using the challenges faced by the characters and the world as a means to teach the fundamentals of programming logic. Figure 2 shows an example of a natural language as implemented in the game.

Figure 2 – Natural Language Mechanic in Game

On the maps, it is possible to identify where a mission will begin through dialogue bubbles that appear above the NPCs' heads. Once the mission is completed, these bubbles become invisible, providing a visual indicator of game progress. As shown in Figure 3.

Figure 3 – Quest Indicator on Map

Figure 4 illustrates how the missions are structured. A problem is presented, and solving it involves creating an algorithm. The player must analyze the requirements of the problem and select, from the available options, the response that meets all the necessary specifications to resolve the NPC's situation. If the player makes a mistake, they can try as many times as needed until all answers are correct, with no penalty for errors made.

Figure 4 – Question Mechanic During a Mission

If the player answers a question incorrectly, a dialogue appears informing them that they did not obtain the scroll and can try again at a later time, as shown in Figure 5.

Figure 5 – Display of a Failed Interaction During a Quest

When the player answers all questions correctly, the dialogue continues, accompanied by a visual indicator that signals the player's success in the mission. Additionally, if the player initiates a dialogue with the NPC after completing the mission, the NPC will thank the player for resolving their problem, Figure 6.

To represent player dialogues and interactions on RPG Maker maps, Events are used. Through these, developers can set up conversations, object interactions, quest systems, and more. Events can be categorized based on their functionalities.

There are primary events, which represent essential conversations and interactions necessary for game progression, and secondary events, which are used for controlling switches, transitioning between maps, animations, etc. Figure 7 below highlights examples of the events mentioned.

Figure 6 – Display of a Successful Interaction

Figure 7 – Overview of Events on the City Entrance Map

Figure 8 below describes the main event creations for the quest involving the character Alberic Fizzlepot. It presents the key events used in constructing the mission, including primary events highlighted in blue, animations used in the game setting in purple, and player movement events across the map marked in yellow.

Figure 8 – Overview of Events for the Quest of the Character Alberic Fizzlepot

To assist beginner players in programming, a book was created as a reference guide, allowing players to review all the content presented from the start of the game to the final mission. Each page focuses on a specific topic the player has learned throughout the various maps. The book can be accessed at any time by pressing the [TAB] key on the keyboard. It covers topics introduced in programming logic, such as data input/output, processing, variables, decision structures, and repetition structures. Figure 9 below illustrates content related to the topic of Inputs, accompanied by an example in pseudocode.

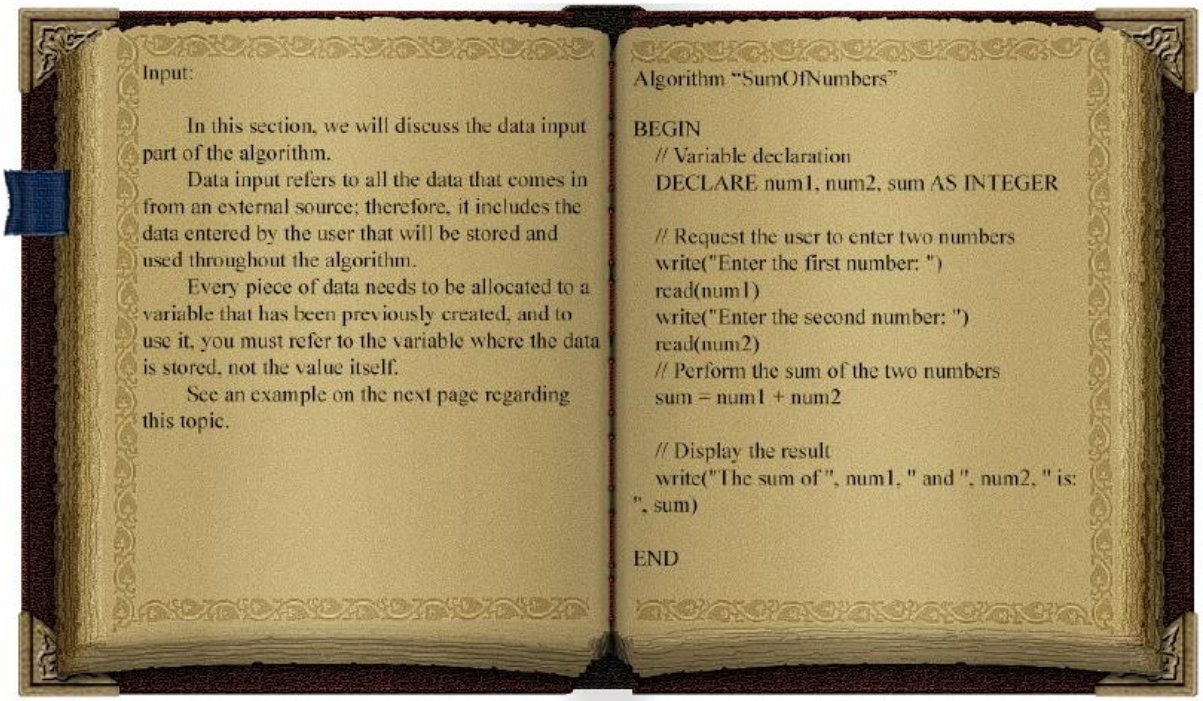

Figure 9 – Overview of Book Pages for Player Assistance: Data Input

Figure 10 below presents new content, this time about Loops. This is the last page of the player's help book and includes a brief description of what loops are and the different types of loops, accompanied by an example in pseudocode.

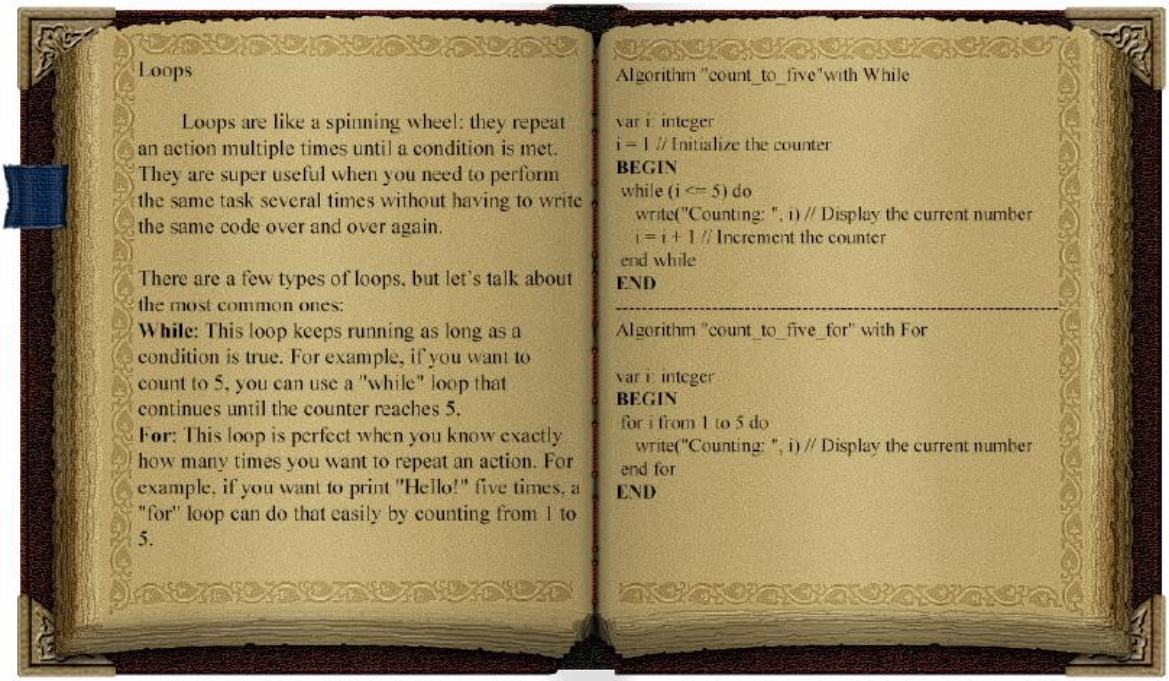

Figure 10 – Overview of Book Pages for Player Assistance: Loops

Another book was created containing instructions on the commands that players will use in the game, providing information on the actions of each button without disrupting the player's immersion, Figure 11.

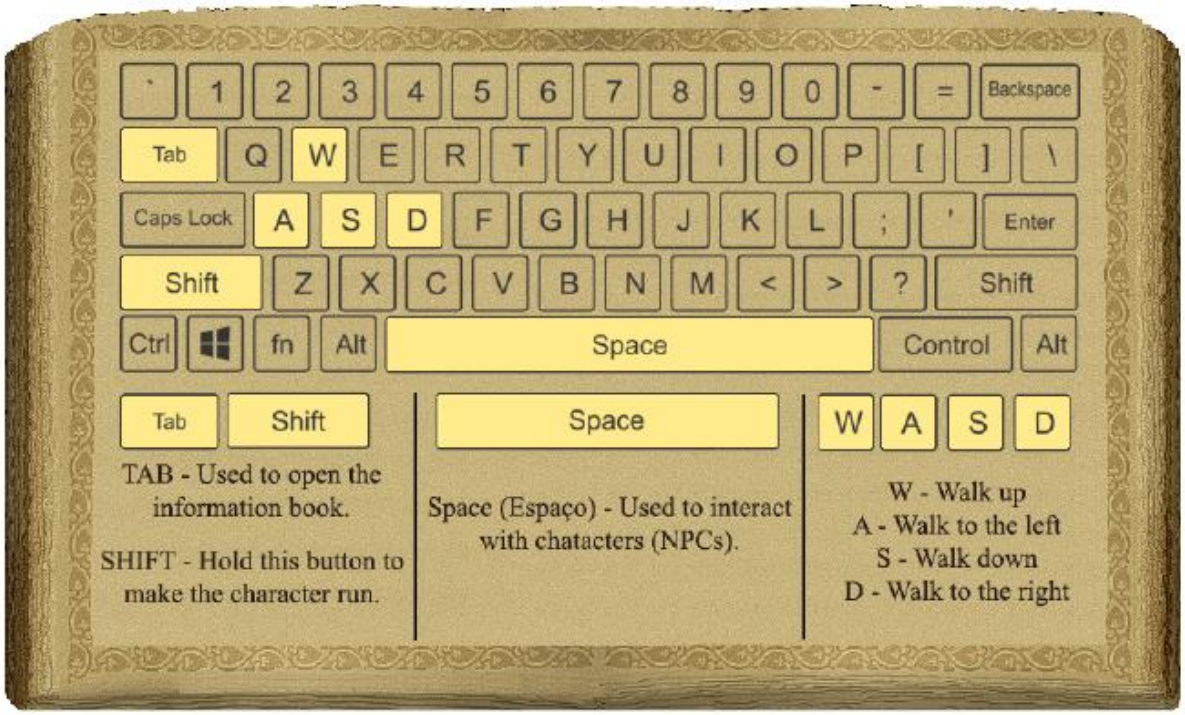

Figure 11 – Instruction Book Screen

After development is complete, or if you want to create a test version of the game without using the engine for testing, it is possible to generate an executable for the desired platform. This allows for testing without exposing the scripts used and enables a quick and simple release on multiple platforms without the need to rewrite code, saving development time.

IV. DISCUSSION AND CONCLUSION

The creation of a serious game for teaching programming logic has proven to be a promising approach to addressing retention and dropout challenges in introductory technology courses. The game provides an immersive setting that engages students and facilitates learning, promoting an interactive and engaging experience that connects them to programming concepts in a practical way. This approach stands out for its non-linear learning, allowing students who need more time to advance at their own pace, revisiting content without being confined to a rigid sequence. This feature is particularly valuable for students who need additional practice on specific topics before moving forward.

The game mechanics encourage practice and reinforcement of logic concepts, enabling students to return to the game and consolidate their knowledge through repetition and continuous practice, which studies show is essential for retaining content in technical areas. Another strength of the project is the ability to compile the executable for different platforms, expanding its reach and facilitating access for students using various operating systems.

In this way, the developed game is an accessible and effective tool that can support new students in learning programming logic, promoting a stronger foundation and contributing to reducing dropout and retention rates in technology courses.

REFERENCES

[1]. BAIST, A.; PAMUNGKAS, A. S. Analysis of student difficulties in computer programming. Volt: JurnalIlmiah Pendidikan Teknik Elektro, v. 2, p. 81–92, 2017. 2

[2]. BARTON, M.; STACKS, S. Dungeons and desktops: The history of computer role-playing games 2e. [S.l.]: AK Peters/CRC Press, 2019. 10

[3]. CASERMAN, P. et al. Quality criteria for serious games: Serious part, game part, and balance. JMIR Serious Games, v. 8, 2020. 8

[4]. CORMEN, T. H. et al. Introduction to algorithms. [S.l.]: MIT press, 2022. 7 DEMIRKIRAN, M.; HOCANIN, F. T. An investigation on primary school students' dispositions towards programming with game-based learning. Education and Information Technologies, v. 26, p. 3871 – 3892, 2021. 1

[5]. FILHO, C. B. P.; ALBUQUERQUE, R. M. de. Uma análise da história dos rpgs (roleplaying games) de mesa brasileiros. SBGAMES, v. 17, p. 29, 2018. 10 [In Portuguese]
[6]. FILHO, R. S. de S. et al. Análises do índice de reprovação na disciplina de tecnologia da informação i da universidade federal de juiz de fora. 2018. 8 [In Portuguese]
[7]. JUNIOR, O. V. d. S. et al. Predição do rendimento dos alunos em lógica de programação com base no desempenho das disciplinas do primeiro período do curso de ciências e tecnologia utilizando técnicas de mineração de dados. BrazilianJournalOfDevelopment, 2020. 8 [In Portuguese]
[8]. MORATORI, P. B. Por que utilizar jogos educativos no processo de ensino aprendizagem. UFRJ. Rio de Janeiro, v. 4, 2003. 1, 4 [In Portuguese]
[9]. MOTTA, R. L.; JUNIOR, J. T. Short game design document (sgdd). Proceedings of SBGames, v. 2013, p. 115–121, 2013. 9
[10]. PAUL, P. S.; GOON, S.; BHATTACHARYA, A. History and comparative study of modern game engines. International Journal of Advanced Computed and Mathematical Sciences, v. 3, n. 2, p. 245–249, 2012. 9
[11]. ROCHA, A. S. et al. Utilização do scratch como ferramenta de auxílio à aprendizagem de programação. In: Anais do COBENGE 2013-XLI Congresso Brasileiro de Educação em Engenharia, Gramado, RS, Brasil. [S.l.: s.n.], 2013. 8 [In Portuguese]
[12]. SALOMÃO, J. S., dos SANTOS, C. E., GIANCOLI, A. P. M., & AMATE, F. C. (2017). Development of a Serious Game to Assist in Teaching History. In 2nd International Conference on Wireless Communication and Network Engineering (WCNE 2017) (pp. 474-479).
[13]. SCHERER, D.; BATISTA, D. V.; MENDES, A. de C. Análise da evolução de engines de jogos. In: SBC. Anais do V Congresso sobre Tecnologias na Educação. [S.l.], 2020. p. 425–434. 9 [In Portuguese]
[14]. SOUZA, M. d.; FRANÇA, A. C. C. Um estudo sobre as dificuldades no processo de aprendizagem de programação no curso de análise e desenvolvimento de sistemas na fafica–faculdade de filosofia, ciências e letras de caruaru-pe. Revista da Escola Regional de Informática, v. 2, n. 2, p. 19–27, 2013. 4 [In Portuguese]
[15]. WIEGERS, K. E.; BEATTY, J. Software requirements. [S.l.]: Pearson Education, 2013. 14
[16]. YUSOFF, A. et al. A conceptual framework for serious games. 07 2009. 9