UNIVERSITY TEKNOLOGI MARA

CODE[X]: NEXUS, A NARRATIVE-DRIVEN 2D TOP-DOWN STYLE ADVANTURE GAME ABOUT INTRODUCTORY-LEVEL C++ PROGRAMMING USING GODOT 4

AMIR HAFIZI BIN MUSA

BACHELOR OF COMPUTER SCIENCE (HONS.)

JANUARY 2026

Universiti Teknologi MARA

Code[X]: Nexus, a Narrative-Driven 2D Top-Down Style Adventure Game about C++ Programming using Godot 4

Amir Hafizi Bin Musa

Thesis submitted in fulfillment for
Bachelor of Computer Science (Hons.)
Faculty of Computer and Mathematical Sciences

JANUARY 2026

SUPERVISOR'S APPROVAL

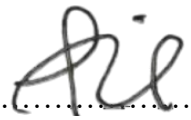
CODE[X]: NEXUS, A NARRATIVE-DRIVEN 2D TOP-DOWN STYLE ADVENTURE GAME ABOUT C++ INTRODUCTORY-LEVEL PROGRAMMING USING GODOT 4

By

AMIR HAFIZI BIN MUSA
2024745815

This thesis was prepared under the direction of thesis supervisor, Ahmad Farid Bin Najmuddin. It was submitted to the Faculty of Computer and Mathematical Sciences and was accepted in partial fulfilment of the requirements for the degree of Bachelor of Computer Science (Hons.).

Approved by:

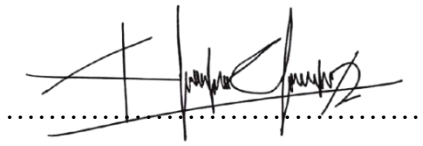


……
Ahmad Farid Bin Najmuddin
Thesis Supervisor

JANUARY 28, 2026

DECLARATION

I certify that this report and the research to which it refers are the product of my own work and that any ideas or quotation from the work of other people, published or otherwise are fully acknowledged in accordance with the standard referring practices of the discipline.



……

AMIR HAFIZI BIN MUSA

2024745815

JANUARY 28, 2026

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CBOT	C++ inspired language
CS	Computer Science
GBL	Game-Based Learning
GDLC	Game Development Life Cycle
HRM	Human Resource Machine
HUD	Head-Up Display
MEEGA+	Multidimensional Evaluation of Educational Games Plus
NPC	Non-Player Character
RPGs	Role-Playing Games
SMEs	Subject Matter Experts
UI	User Interface
UX	User Experience

ACKNOWLEDGEMENT

In the name of Allah, the Most Gracious, the Most Merciful. First and foremost, I would like to express my deepest gratitude to Allah S.W.T. for giving me the strength, patience and good health to complete this Final Year Project, Code[X]: Nexus, within the given time. This journey has been challenging and His blessings have been my source of perseverance. Secondly, I would like to thank my project supervisor, Mr. Ahmad Farid Bin Najmuddin, for the immense value he brought to my learning experience by providing positive criticism and encouragement that assisted me to develop the Code[X] project. He provided an enormous amount of knowledge regarding Game-Based Learning which directed the path of this research. Additionally, the theories that are embedded within Game-Based Learning were implemented by him as well.

Lastly, I would like to thank the Faculty of Computer and Mathematical Sciences at Universiti Teknologi MARA (UiTM) for allowing me to utilize their resources to create a conducive academic environment for this research. My heartfelt thanks also go to my dear parents and family who have shown unrelenting love, prayed continuously and financially supported me throughout my degree program. It is your unconditional belief in my ability that motivated me to continue on during the most trying periods of this degree program.

A special acknowledgement goes to my fellow coursemates and friends who stood by me, shared ideas, and provided moral support. I would also like to thank the 31 respondents who voluntarily participated in the User Acceptance Testing (UAT) phase. Your feedback and honesty were crucial in refining the gameplay and proving the educational effectiveness of this project. Finally, to everyone who contributed directly or indirectly to the success of Code[X]: Nexus, thank you. Amir Hafizi Bin Musa Bachelor of Computer Science (Hons.) Universiti Teknologi MARA (UiTM) July 2025.

ABSTRACT

Learning introductory programming, especially C++ is difficult for new programmers due to the abstractness of the syntax and logic involved. Traditional teaching methods have difficulty providing students the instant visual feedback they require to remain engaged and motivated in their studies which contributes to the high dropout rate of students enrolled in Computer Science programs. Therefore, this final year project will be presented with a goal on developing a narrative-driven 2D top-down adventure game called Code[X]: Nexus. This project is designed in becoming a supplemental learning tool for the computer science curriculum in the Diploma program at Universiti Teknologi Mara. Code[X]: Nexus was created by utilizing the Godot Engine 4 as the game engine. The game utilizes a Game Based Learning (GBL) methodology that uses visualizations of abstract concepts in the form of game mechanics. A "compiler puzzle" was also developed to allow players to write and debug code to solve puzzles and continue their way through a science fiction narrative. The use of "scaffolding" was integrated into the design of the game to introduce complex topics including Variables, Loops and Conditionals to players in an increasingly difficult manner. Functional testing and User Acceptance Testing (UAT) were conducted on Code[X]: Nexus with 31 participants who are part of the target demographic. The results of the UAT demonstrated that the game improved the confidence of the participants with respect to the completion of the puzzles as 90.3 % of the participants indicated that the interactive puzzles were more effective than traditional textbooks. In addition, 67.7% of participants preferred the game-based method of instruction versus standard lecture methods. Overall, it can be concluded that the use of narrative driven gamification is an effective pedagogical strategy to bridge the gap between theory and practice in technical education.

CHAPTER 1

INTRODUCTION

1.1 Introduction

In today's world, the processes of learning and teaching are transforming at a rapid rate. Conventional methods such as books and lectures are often insufficient to engage the interest of most students, particularly when it comes to complex subjects like Computer Science (Al-Maroof et al., 2024). With technology becoming deeply integrated into our lives, there is an increasing demand for innovative and efficient ways to learn, one of which is Game-Based Learning (GBL) (Ishaq et al., 2024).

GBL exploits the playfulness and interactive nature of computer games to present learning concepts in an easy and enjoyable manner. This approach can simplify learning, making it more enjoyable and facilitating an easier understanding of abstract ideas that otherwise cause problems (de Oliveira, de Oliveira, & de Souza, 2022). However, while GBL in Computer Science has the potential to reduce or eliminate study difficulties, it is still not as commonly utilized as it could be in learning games designed for Computer Science students (Kalogiannakis, Papadakis, & Zourmpakis, 2021).

Briefly, this study will examine how computer games able to aid students in Computer Science field by giving them a better, interactive way of learning while contributing to the growing area of educational technology.

1.2 Background of Study

Nowadays, the world is witnessing significant and rapid changes in technology, which are impacting all aspects of life, including education. With the prevalence of digital media, online environments, and interactive technologies, the effectiveness of traditional classroom learning in the 21st century is being called into question. While textbook and lecture-based learning have been central to formal education for centuries, contemporary students are increasingly drawn towards interactive, graphical, and game-like modes of engagement (Kalogiannakis et al., 2021).

In recent years, educational studies have highlighted a widening gap between traditional teaching methods and the engagement levels of students, particularly in demanding subjects like Computer Science, History, and Mathematics (Al-Maroof et al., 2024; Tan, Neill, & D'Souza, 2021). Such subjects often involve sequential concepts or abstract ideas that can be difficult to grasp and retain in passive learning environments. Traditional lectures, which are often non-interactive and lack modern storytelling elements, can leave students feeling bored, demotivated, or confused (Chan et al., 2023). This has led to the increased adoption of technology in education, particularly in the form of Game-Based Learning (GBL)

GBL is the application of video games and their mechanics to teach specific skills or knowledge domains in an interactive and engaging manner. Current research suggests that learning games have the potential to significantly improve memory recall, problem-solving abilities, and the motivation to learn (de Oliveira, de Oliveira, & de Souza, 2022). By combining a story with interactive challenges, learning games offer a dynamic alternative to standard educational methods. Among different game genres, 2D top-down style adventure games are ideally suited for educational experiences because they provide a linear progression that allows developers to seamlessly incorporate puzzles, narratives, and question-based challenges (van Gaalen et al., 2021). If designed with care, such games can become effective instruments for reinforcing learning goals while keeping the player interested and curious. Yet, despite the established advantages of gamification in education, many institutions continue to

rely on static content delivery (Tian et al., 2024). This disconnects contemporary learning preferences and conventional educational tools presents an opportunity to investigate new methods that can improve student learning outcomes.

Thus, this study proposes the development of the project, a 2D Learn & Gain top-down style adventure video game based on the Godot Engine (v4.4), for university students studying Computer Science. Interactive puzzle, story, and gamified obstacle perhaps will help the students to have a more engrossing, stimulating, and effective learning experience that meets the cognitive and motivational needs of the modern learner.

1.3 Problem Statement

As the world of technology expands, the schooling system needs better ways of teaching because some of them still use passive methods of teaching like lectures. Traditional ways of teaching like lectures, textbooks, and slide presentations remain, but these methods do not necessarily engage students actively or support knowledge retention, especially in technical fields like Computer Science (Chan et al., 2023).

Most learners, especially the first-year diploma students of university, are struggling to be interested in Computer Science (Tan, et al., 2021; Holly, et al., 2024). This is based on the reality that the subject is very analytical and abstract, and is often delivered in non-interactive and static approaches that are criticized for being too theoretical (Al-Maroof et al., 2024; Holly et al., 2024). There is also a significant gap in technology that facilitates individuals to learn in a fun way, for example through games and stories (Cheong, Flippou & France, 2020). When there is no context, problem-solving activities, and stories, students may struggle to relate theory to real-life situations, which can lead to lower motivation and performance (de Oliveira, de Oliveira, & de Souza, 2022).

Game-Based Learning (GBL) has been one such remedy for these issues by the incorporation of learning content with engaging mechanics that encourage analytical thinking, collaboration, and exploration (Kalogiannakis, et al., 2021). However, though gamified learning solutions have gained growing popularity, most available solutions are either too simple or not exactly tailored to the requirements of the Computer Science curriculum (Tan, et al., 2021). In addition, students' varied learning needs demand experiences that are not only interactive, but narrative, image-centered and pedagogically designed.

The study by Kucher, T. (2021) conclude that games that have definite stories, decisions to proceed and associated challenges assisted students in recalling procedural and theoretical information more effectively than normal tests. No such tools are being created in tertiary education, particularly for Southeast Asian Computer Science students (Kucher, T., 2021).

To summarize, students struggle to remain engaged and retain information when learning Computer Science through traditional methods like lectures and textbooks. Existing game-based learning software often fails to address this, being either too broad or lacking the specific, structured content required for the subject (Tian et al., 2024). Consequently, there is a scarcity of interactive, story-driven educational games that are specifically mapped to the curriculum of first-year university diploma courses in computer science. This continuous lack of innovative tools negatively impacts students' motivation, engagement, and overall performance in their technical courses.

1.4 Project Objectives

To address the problem of student motivation and knowledge retention in Computer Science learning, this project aims to design a 2D Learn & Gain Top-down Style Adventure Game for first-year university diploma students to extend learning by using interactive storytelling and lesson gamification. In this respect, objectives are to:

1. To identify an effective game-based learning (GBL) techniques in programming education in enhance student motivation.
2. To develop a 2D top-down style learning adventure game, in learning programming concepts that includes GBL techniques using Godot Engine v4.4.
3. To assess the usability of the developed GBL game towards the targeted students.

1.5 Scope of Study

The project will focus on the implementation of Game-Based Learning (GBL) in Computer Science focusing on programming topics education through a 2D top-down style adventure game. The project aims to explore on how interactive narrative and puzzle goes within a top-down game can helps students' interest, improve concept comprehension, and improve memory recall, especially among university students. The research investigates on the way of how study content can be more gamified and placed within a context of a narrative. This integrates learning with advancement in a game in overcoming common problems related to the traditional means of learning.

The proposed solution of the project is able to provide:

- A narrative-driven gameplay experience with progression themed levels representing variety of fundamental topics in C++ programming field.
- A challenge-based system that enables students lean and progress by answering questions or solving puzzles in relation to the subject field.

- A visual cues and storytelling elements designed to enhance and improve students understanding regarding the concept topics such as history and overview of C++, basic syntax and data types, basic data structures and basic algorithms.

This project will be developed using the Godot v4.4 game engine, it will be targeted towards the Windows and Web (HTML5) platforms for deployment. The game is designed to be lightweight, has low system specification requirements and easily accessible via browser or Windows executable file (.exe), while also suitable for both classroom use or even independent learning. The target users are first-year of diploma in Computer Science of UiTM who are new to Computer Science or are struggling with the foundation concept.

The scope of the game's functionality revolves around a single-player, 2D top-down RPG-style adventure with level-based progression. It is designed to offer a narrative-driven gameplay experience where levels are themed to represent fundamental topics in the C++ programming language. The core learning mechanic is a challenge-based system that allows students to progress by answering questions or solving puzzles related to their studies. This is supported by visual cues, storytelling elements, assets, and animations designed to enhance the learning themes. The game will also feature basic performance tracking, monitoring metrics such as completed questions and total playtime.

The project is intentionally restricted to introductory-level programming topics. Specifically, the educational content will cover the history and overview of C++, basic syntax and data types, basic data structures, and basic algorithms. The target users are specifically first-year Diploma in Computer Science students at UiTM who are either new to the field or struggling with foundational concepts. This project will be developed using the Godot Engine v4.4 and will be targeted for deployment on Windows and Web (HTML5) platforms.

1.6 Significant of Study

With the presented solutions, the significant towards the body of knowledge lies within the utilization of Game-Based Learning (GBL) principles in Computer Science education, more precisely the learning of abstract and procedural material by students. This will help in emerging the area of educational technology and gamification, showing how narrative-based game environments can enhance learning experiences and retention rates among students in the related field.

The significance of this study is twofold, first it offers a practical model for the development of learning games for topic-specific content towards other game developers and it presents the argument for the value of interactivity, narrative, and challenge-based progression while fostering student motivation and understanding. In doing so, it helps bridge the gap between abstract theory and actual practice in an interesting and easily accessible way.

This study is primarily useful for first-year university diploma computer science students who struggle with traditional learning, offering them an entertaining and enjoyable alternative that transforms difficult programming concepts into a more easily digestible format. It also benefits lecturers, who can utilize the resulting game as an alternative teaching tool to enhance their lessons, encourage critical thinking, and introduce variation in classwork. Ultimately, the study aims to provide a convenient learning environment where students can learn at any time and from anywhere, thereby promoting self-learning.

Through the integration of narrative and subject-based challenges, the project demonstrates the ways in which purposeful game design can positively enhance the practice of education by promoting more effective creation and accessible learning software in the digital era where we must adapt, not reject.

1.7 Summary

This chapter describes that Computer Science education must be enhanced by innovative and interactive approaches like Game-Based Learning (GBL). Conventional teaching cannot hold students' attention span and interest, particularly if the subject matter is complicated or abstract in nature, such as in the case of algorithms and binary logics.

Because of this, in order to address this issue, the proposed project will provide interactive and narrative learning experiences. The purpose of this proposal is to enhance student engagement and retention by incorporating educational materials within a gamified system, developed using the Godot Engine v4.4.

The solution scope involves targeting first-year university diploma students with an emphasis on programming courses at the introductory level through single-player mode with puzzles and quizzes that are dialogue-based. The study significance involves the contribution to educational technology through the demonstration that storytelling, game design, and visual learning can come together to work effectively in augmenting and supplementing the current model of formal education.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter provides an overview of current research regarding Game-Based Learning (GBL) and its implementation in Computer Science education, specifically in the instruction of introductory programming. This is to lay down a theoretical background for this project, an interactive 2D top-down adventure game created with the Godot Engine as an instruction medium of C++ programming for first-year diploma Computer Science students.

Classical teaching methods lead to disengagement among students since programming concepts are intangible and decontextualized (Tan et al., 2021). Educators have subsequently looked for substitute interactive and engaging learning tools, like games, that promote active learning, problem-solving and engagement through gameplay mechanics including progression mechanisms, challenges and story (Qian & Clark, 2019; Cheong et al., 2020; Borrás-Gené et al., 2024).

This literature review aims to explore relevant GBL approaches in education program games such as Colobot: Gold Edition, CodeCombat and Human Resource Machine that identifies elements which enabling effective learning process. These will be utilized to inform design within this project in a way that considers the integration of actual programming tasks in an interactive story environment that is appropriate for new students (Kucher, 2021). Figure 2.1 shows the overall tree structure of the proposed project.

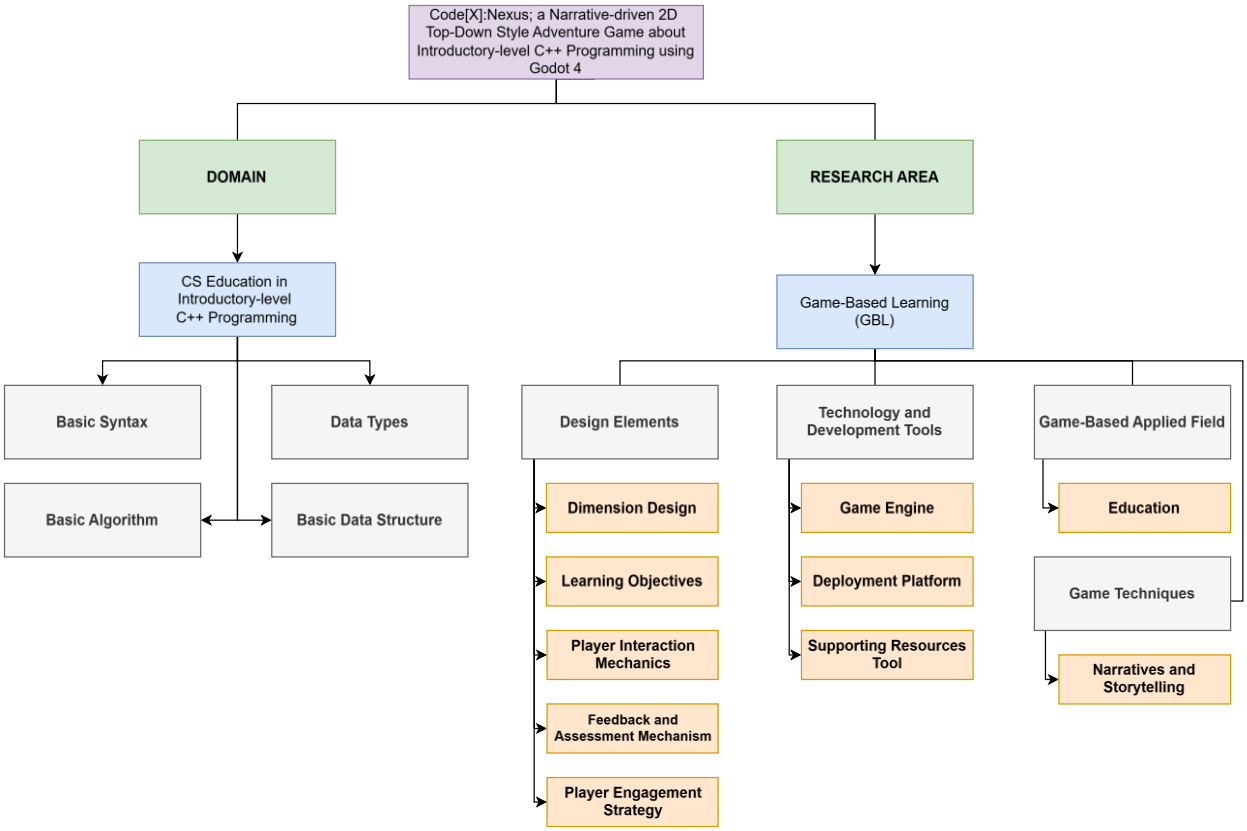


Figure 2.1 Tree Structure of the Proposed Project

2.2 Overview of Computer Science Education

Computer Science (CS) education has become a widely viewed technical education in the modern era, particularly at diploma and undergraduate levels. Nevertheless, it is documented that many freshman CS students struggle with fundamental programming concepts, most notably due to the fact that subjects such as variables, loops, conditionals and object-oriented design are often abstract in nature (Papastergiou, 2009; Holly et al., 2024).

Traditional education heavily relies on methods like lectures, notes, slides, textbook and coding exercises, which has shown often fail to fully engage students or provide meaningful context for learning process (Tan et al., 2021). This causes the drive in having low motivation, poor retention rates, low attention span and high dropout rates especially among first-year students who are just completely new to programming (Arnedo & García, 2023).

Studies have determined that the majority of students consider programming as difficult and out of reach, primarily because ideas are taught in isolation without clear real-world applications (Cheong et al., 2020). In addition, the lack of immediate feedback and interactivity in conventional settings limits channels for active learning, trial and error, which required to gain mastery in programming field (Qian & Clark, 2019).

To counter these challenges, educators and research professionals have resorted to alternative learning strategies, including interactive tools, visual programming environments and gamified learning systems (Kucher, 2021; Tene et al., 2025). All these are designed to increase access to programming by positioning learning in immersive and contextualized experiences.

One of the most promising areas is the integration of Game-Based Learning (GBL) into the teaching of CS. Learning games are structured but enjoyable environments in which students can learn by trying out, problem-solving and gradually developing mastery over concepts and representations of intellectual processes employed in actual programming (van Gaalen et al., 2021). Examples of games such as Colobot: Gold

Edition, CodeCombat and Human Resource Machine showcase how learning goals can be transferred to game mechanics to improve participation and comprehension (Heithausen, 2020; Kunovic et al., 2024).

With the growing demand for skilled programmers and the importance of early success in shaping students' confidence in CS, there is a strong need to explore new methods, which aligned with today's digitally native students' demands and learning patterns (Boyle et al., 2016).

2.3 Overview of Game-Based Learning (GBL)

Game-Based Learning (GBL) is the practice of using digital games in learning contexts to facilitate improved learning outcomes through an interactive and dynamic experience (Kalogiannakis, et al., 2021). While traditional methods often rely on passive learning approaches such as reading books and attending lectures, GBL promotes active engagement by allowing students to tackle problems, receive immediate feedback, and progress at their own pace (de Oliveira, et al., 2022).

The theoretical foundation of GBL is closely linked to experiential learning, which posits that learning occurs most effectively when students engage in a concrete experience, followed by reflection and application (Lozano-Lozano et al., 2023; Cheong et al., 2020). Computer games inherently support this cycle by immersing players in challenges that require decision-making, experimentation, and iterative refinement. Research suggests that games are effective because they align with the learning patterns of contemporary students, who are accustomed to fast-paced, visually rich, and interactive digital environments (Chan et al., 2023).

Because of this, it increases the motivation for them, which is a key factor in student engagement and long-term knowledge retention (Holly et al., 2024). Furthermore, games can also reduce cognitive load by presenting information in manageable chunks, allowing the students to build confidence gradually (Choi & Choi, 2024, Herlambang et al., 2024).

GBL has shown promises in helping students grasp abstract concepts like loops, conditionals, functions and object-oriented programming (Tan et al., 2021). Existing GBL games like Colobot: Gold Edition, CodeCombat and Human Resource Machine have proved that engaging world building and well-crafted gameplay mechanics, including progression systems, meaningful challenges and immediate feedback significantly enhance learning while improving understanding of programming basics (Arnedo & García, 2023; Heithausen, 2020).

Even though GBL will not replace traditional teaching practices, it can be used as a tool that enhances motivation, promotes self-learning and supports experimentation, which are key processes in acquiring programming skills (van Gaalen et al., 2021).

These insights lay the foundation for the proposed project, which integrates GBL elements into a narrative-driven 2D top-down adventure game designed in teaching introductory-level C++ programming to first-year diploma students.

2.4 Specific Description of Game-Based Learning

Game-Based Learning (GBL) goes beyond entertainment by integrating structured learning objectives into gameplay, which promotes active engagement rather than passive reception (Kalogiannakis et al., 2021). Games act as interactive systems that can simulate real-world challenges, promote decision-making, and offer immediate feedback for key elements that enhance deep learning and retention (Cheong et al., 2020).

According to recent studies, well-designed educational games boost student motivation and understanding, particularly for complex or abstract topics like programming logic and syntax (Al-Maroof et al., 2024). This aligns with experiential learning theory, which posits that meaningful learning happens through hands-on experiences, reflection, and application. Digital games naturally facilitate this process by immersing players in structured challenges that involve problem-solving, trial and error, and iterative improvement, effectively mirroring real-world programming tasks (Lozano-Lozano et al., 2023).

In GBL environments, progressive difficulty, embedded tutorials and storytelling help students build confidence and competence gradually (van Gaalen et al., 2021). Games like Colobot: Gold Edition, CodeCombat and Human Resource Machine align gameplay with learning goals to boost engagement and understanding (Heithausen, 2020; Arnedo & García, 2023). They introduce programming concepts step-by-step, starting with basic commands, then advancing to loops, conditionals, and functions (Heithausen, 2020).

GBL supports self-directed learning, letting students explore content at their own pace and revisit difficult areas without pressure (Kucher, 2021). It also enables mastery-based progression, requiring a full understanding of one concept before advancing, which reinforces fundamentals before introducing greater complexity (Zainuddin at el., 2020).

In introductory-level programming, GBL offers a low-stakes environment where mistakes are part of learning, not a failure (Choi & Choi, 2024). This is crucial for first-year diploma students lacking confidence in technical subjects. By presenting programming as gameplay, students become more open to experimenting, debugging, and refining code, concluding the key skills in software development.

2.5 Techniques in Game-Based Learning

Game-Based Learning (GBL) relies on a variety of engagement-enhancing techniques that promote motivation, knowledge retention and problem-solving skills (Di Nardo et al., 2024). Among the most impactful are progression systems, challenge-based puzzles and narrative-driven storytelling (van Gaalen et al., 2021). These techniques are particularly effective in educational contexts such as introductory programming instruction, where abstract concepts can be difficult for students to grasp without meaningful context or structured support (Al-Maroof et al., 2024).

2.5.1 Progression System

A progression system is a game design mechanism that allows players to advance through increasing levels of difficulty, often unlocking new abilities, tools, or content along the way (Cheong et al., 2020). In educational settings, progression systems help enhances learning by gradually introducing new challenges while reinforcing previously acquired knowledge (Zainuddin et al., 2020). This approach aligns with mastery-based learning, where students must demonstrate understanding before moving to more complex topics.

In programming education, this structure mirrors how coding concepts are typically introduced, starting with basic syntax and logic before progressing to loops, conditionals, functions, and object-oriented principles (Arnedo & García, 2023). Games like Colobot: Gold Edition and CodeCombat use level-based progression to guide players through increasingly complex programming tasks, making them ideal models for this project (Heithausen, 2020). Figure 2.2 show the example usage of leveling system as the main progression system.

Figure 2.2 Examples of using leveling up and stage levels selection as the game's core progression system.

2.5.2 Challenge and Quest

Challenge-based gameplay involves presenting students with problems or quests that require logical reasoning, decision-making, and iterative trial-and-error (Kucher, 2021). Puzzles are one of the most commonly used challenge formats in educational games, especially those designed for teaching programming fundamentals (Heithausen, 2020).

Games like Human Resource Machine present players with input-output puzzles that simulate real-world programming problems (e.g., sorting values, comparing numbers, managing memory cells), encouraging algorithmic thinking and debugging skills (van Gaalen et al., 2021). These mechanics not only reinforce technical knowledge but also foster self-directed exploration, which is essential for mastering complex subjects like programming (Choi & Choi, 2024). Figure 2.3 shows the usage of challenges system, meanwhile figure 2.4 shows the example of quest system.

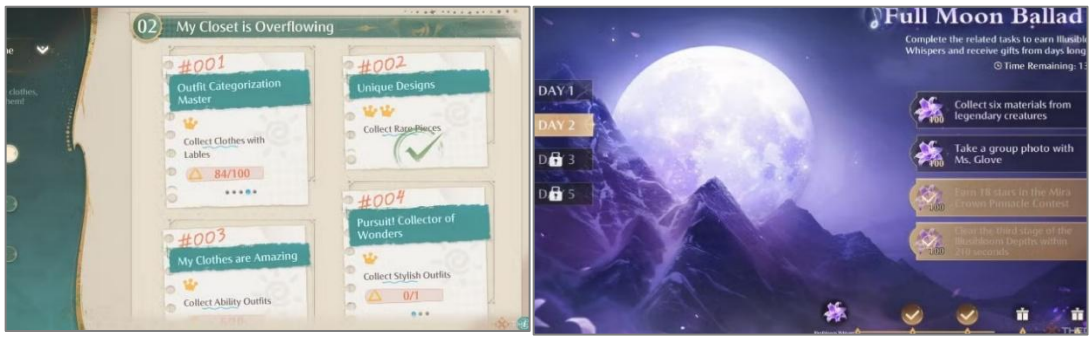


Figure 2.3 Examples of games using challenges to immerse players into the game world

Figure 2.4 Examples of games using quest system to give players goals to pursue

2.5.3 Narratives and Storytelling

Narrative elements provide emotional engagement and contextual meaning to gameplay, making them powerful tools for enhancing motivation and memory recall (Kucher, 2021). In educational games, stories help frame abstract problems within relatable scenarios, allowing players to better understand the relevance and application of what they're learning (Cheong et al., 2020).

For example, Colobot: Gold Edition immerses players in a science fiction storyline where they must program robots to colonize alien planets using a C++ inspired language called CBOT (Alves & Letouze, 2018). This narrative-driven approach increases student investment and encourages deeper cognitive processing of programming concepts (Giannakoulas & Xinogalos, 2024).

While some educational games like CodeCombat focus more on mechanics than story, research suggests that adding strong narrative components significantly enhances both emotional engagement and long-term interest in learning (van Gaalen et al., 2021). This shapes the proposed project's design, uniting all three techniques within a cohesive, narrative-driven experience. Figure 2.5 shows the usage of narrative dialogue system, meanwhile figure 2.6 shows the usage of narrative cutscenes.

Figure 2.5 Examples of games using NPC dialogue system for story progression

Figure 2.6 Examples of games using cutscenes to lock players engagement

2.5.4 Techniques Comparison

Table 2.1 compares the Game-Based Learning (GBL) techniques in Colobot: Gold Edition, CodeCombat, and Human Resource Machine. It highlights that while progression systems and challenges are common, strong narrative integration is not always present, informing the design choices for this project.

Table 2.1 GBL Techniques Comparison

	Progression System	Challenge And Quest	Narratives And Storytelling
Definition	Players advance through increasing levels of difficulty, unlocking new content or abilities.	Players solve problems or quests requiring logical reasoning and problem-solving.	Players engage with a storyline that provides emotional context and immersion.
Educational Impact	Promotes mastery-based learning; reinforces prior knowledge before introducing complexity.	Encourages algorithmic thinking, debugging, and iterative learning.	Enhances memory recall, contextual understanding, and emotional investment.
Engagement Level	- High Players feel a sense of achievement as they progress.	- High Challenges stimulate curiosity and persistence.	- Very High Narrative increases emotional connection and long-term interest.
Complexity	Moderate	Moderate to High	High
Relevance to Programming Education	Supports gradual introduction of programming concepts (e.g., syntax, logic, functions).	Reinforces problem-solving and logical reasoning essential for coding.	Helps contextualize abstract concepts within relatable scenarios.
Examples in Educational Games	CodeCombat, Colobot	HRM, CodeCombat	Colobot, HRM, CodeCombat
Limitations	May become repetitive if not paired with variety.	Can frustrate learners if too difficult or unclear.	Time-consuming and may distract from core learning.

2.6 Common Features Related to Proposed Project

Several educational games have been developed over the years to support introductory programming instruction, particularly through Game-Based Learning (GBL) techniques such as progression systems, challenge-based puzzles, and narrative elements (Cheong et al., 2020; Qian & Clark, 2019). Three notable examples that align closely with the proposed project are Colobot: Gold Edition, CodeCombat, Human Resource Machine. Each of these games provides a unique approach in teaching programming concepts and has been used in both formal and informal learning environments. Table 2.2 provides the preview of the game's look.

Table 2.2 Related GBL games preview image

	
Colobot: Gold Edition	CodeCombat
	
Human Resource Machine	

2.6.1 Colobot: Gold Edition

Colobot: Gold Edition is a science fiction-themed game in which players control programmable robots to colonize alien planets using own C++ like language called CBOT (Alves & Letouze, 2018). The gameplay involves solving missions that teach core programming concepts such as loops, conditionals, functions, arrays and object detection.

The game possesses several key strengths, notably its use of a C++ inspired language that makes it highly relevant for students transitioning to real-world programming. Its integration of a strong narrative enhances emotional engagement and motivation, while the gameplay encourages self-directed exploration, problem-solving, and a mastery-based progression. Furthermore, it supports community-driven content creation, which allows educators to extend or customize the game for their specific classroom needs.

However, the game is not without its limitations. The interface, with its 3D environment and complex controls, can be overwhelming for beginners. This issue is compounded by the lack of a structured tutorial system, potentially hindering accessibility for novice students. Additionally, due to its strong technical focus, the game may not appeal equally to all learning styles.

Studies show that Colobot: Gold Edition promotes active learning and enhances understanding of programming logic when integrated into curricula (Alves & Letouze, 2018). Students who engaged with the game reported increased interest in programming and improved confidence in tackling complex tasks. This is because of the comprehensive game design as shown in figure 2.7, 2.8, 2.9 and 2.10.

Figure 2.7 Main menu of Colobot: Gold Edition

This figure shows a main menu screenshot from *Colobot: Gold Edition*. It illustrates how players interact with the UI system of the game.

Figure 2.8 World design concept of Colobot: Gold Edition

This figure displays the world design of the game, which the users are able to fully explore and learn on their own.

Figure 2.9 Mission menu of Colobot: Gold Edition

This figure presents a mission selection system of the game, players will be able to easily select which mission to start on first.

Figure 2.10 Program editor to control the robot mechanism

This figure shows the usage of in-game program editor for players to use in completing the required mission of the game.

2.6.2 CodeCombat

CodeCombat is a browser-based game that teaches real Python and JavaScript syntax by having players write code to solve puzzles and progress through levels (Kroustalli & Xinogalos, 2021). Each level presents a challenge that must be solved using actual programming commands, often involving loops, conditionals, and function calls.

The platform's strengths lie in its practical approach to teaching real coding syntax, making it ideal for students preparing for professional development environments. It offers immediate feedback, which helps students identify and correct mistakes quickly, and includes a gamified progression system that increases motivation and persistence. Furthermore, its effectiveness in reducing cognitive load and improving syntax understanding has been proven.

Nevertheless, the platform has some limitations, including a minimal narrative depth that can limit emotional investment compared to more story-driven games. Some students also found the experience repetitive after completing the core levels, and its limited gameplay variety could potentially affect long-term engagement.

Research indicates that CodeCombat significantly improves students' interest in programming, especially among those who previously found traditional methods unengaging (Kroustalli & Xinogalos, 2021). However, many students expressed willingness to continue only if new challenges or narrative elements were introduced. Figure 2.11, 2.12 , 2.13 and 2.14 shows the overall game structure of this CodeCombat.

Figure 2.11 World selection map inside of CodeCombat

This figure displays the world selection map in CodeCombat, showing themed regions (e.g., "Web Development," "Game Development") with levels marked by progress indicators. Players can select levels to engage in coding challenges.

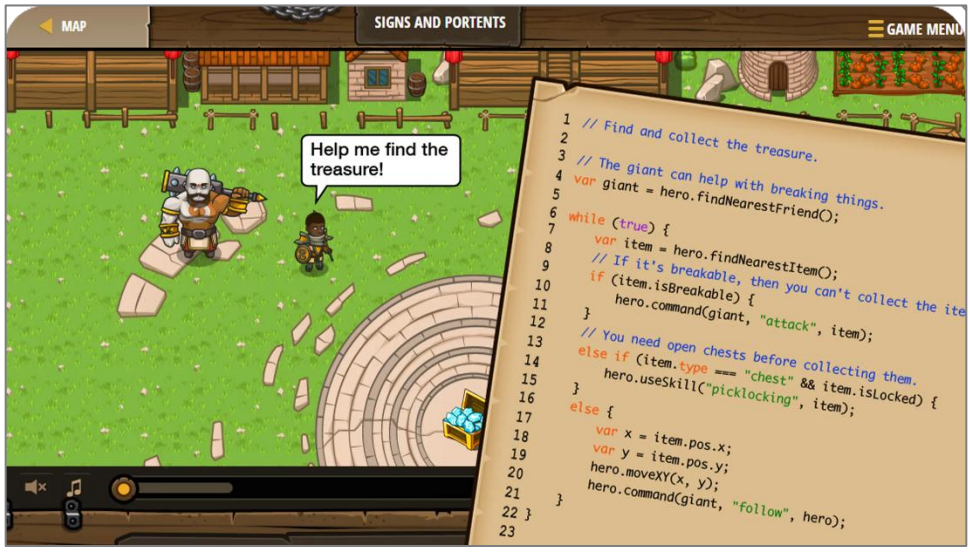


Figure 2.12 One of coding mission inside CodeCombat

This figure illustrates a coding mission in CodeCombat, where players write Python code to solve a puzzle. The interface shows an in-game scene with a character seeking help to find treasure, alongside a code editor displaying syntax for tasks like finding items, checking conditions, and moving characters.

Figure 2.13 Characters menu of CodeCombat

This figure shows the character selection menu in CodeCombat, where players choose their avatar for gameplay. The interface displays a character named "Captain Ana Weston," along with options to select a programming language (e.g., JavaScript) and save their choice.

Figure 2.14 Map exploration inside chosen world

This figure depicts the in-game map exploration screen in CodeCombat, showing a detailed dungeon-like environment with obstacles, pathways, and interactive elements. The interface includes character controls and mission objectives, allowing players to navigate and solve coding challenges within the chosen world.

2.6.3 Human Resource Machine

Human Resource Machine (HRM) is a puzzle-based game that introduces players to algorithmic thinking and basic assembly-level logic using a simplified command set (Heithausen, 2020). Players write programs to move items between input, output conveyors and memory cells, simulating real-world computational tasks.

The platform's strengths are centered on reinforcing essential skills for beginner programmers, such as logical reasoning, debugging, and understanding step-by-step execution. It effectively encourages trial-and-error learning, which supports a deep conceptual understanding, and its simple interface allows for quick immersion without the distraction of visual complexity.

However, it has notable limitations, including a lack of real-code integration that makes it less suitable for students who aim to learn standard programming languages like C++. Additionally, its minimal narrative and visual immersion may reduce long-term engagement, and the platform does not scale well to cover advanced topics or more complex programming structures.

Despite its simplicity, HRM has been praised for its ability to introduce core programming logic in an accessible way (Heithausen, 2020). It supports self-paced learning and encourages optimization through score tracking (e.g., minimizing steps or commands), which motivates players to refine their solutions. Figure 2.15, 2.16, 2.17, and 2.18 shows the great game and world design of for a coding-style game implementation.

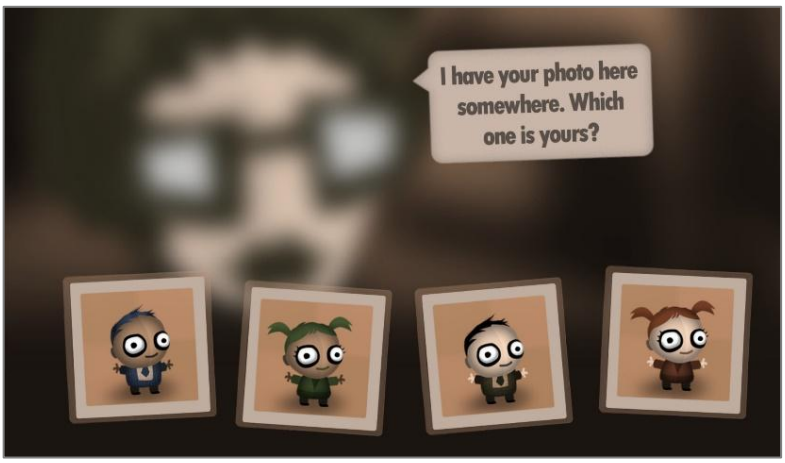


Figure 2.15 Human Resource Machine storyline and dialog system

This figure illustrates the storyline and dialog system in *Human Resource Machine*. It shows a character speaking with a speech bubble that reads, "I have your photo here. Which one is yours?" Below the dialogue, four framed portraits are displayed, suggesting a choice-based interaction or puzzle element central to the game's narrative and gameplay mechanics.

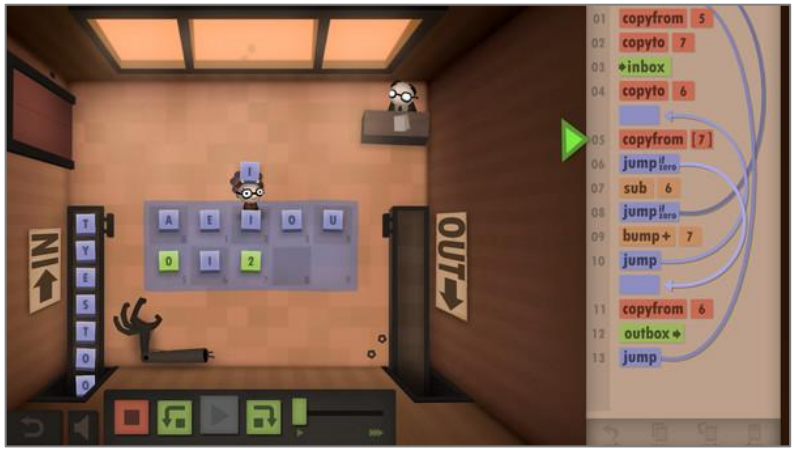


Figure 2.16 Level design of Human Resource Machine stage

This figure depicts the grid-based level design in *Human Resource Machine*, showing a workspace with numbered tiles and commands like copy_from and jump, illustrating the game's procedural programming mechanics.

Figure 2.17 Mission challenges of Human Resource Machine

This figure illustrates a mission challenge in *Human Resource Machine*, displaying a task titled "Sta Challenge" with objectives like using 10 or fewer commands and completing the task quickly. The interface includes options to return or continue, highlighting the game's focus on efficiency and problem-solving.

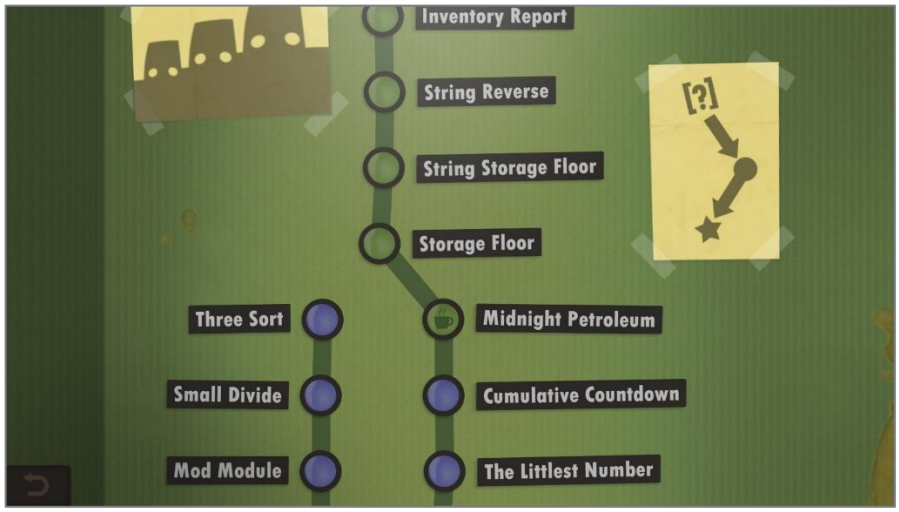


Figure 2.18 Levels progression of Human Resource Machine

This figure shows the levels progression map in *Human Resource Machine*, featuring a network of interconnected stages such as "Inventory Report," "String Reverse," and "Cumulative Countdown." The layout illustrates the structured, branching progression system that guides players through increasingly complex programming challenges.

2.6.4 Features Comparison of Related Game-Based Learning Games

Table 2.3 briefly compares key features of *Colobot*, *CodeCombat*, *Human Resource Machine*, and *Code[X]:Nexus*, focusing on gameplay, programming language, accessibility, target audience, and evaluation approach.

Table 2.3 Comparison to other Related GBL Games

Games / Features	Colobot: Gold Edition	CodeCombat	Human Resource Machine	Code[X]:Nexus (Own Designed Project)
Overview	A 3D sci-fi strategy game where players program robots to solve puzzles and complete missions. Available for download on multiple platforms.	An online browser-based coding game that teaches programming through interactive challenges and quests. Focuses on JavaScript, Python, and other languages.	A visual programming puzzle game set in a corporate office environment. Players use assembly-like commands to automate tasks involving moving data between inbox/outbox and memory spaces.	Interactive, narrative-driven 2D top-down adventure game teaching introductory C++ concepts using Godot Engine v4.4. Deployed on Windows and Web (HTML5).
Characteristics	- Strategy/Adventure genre - Medium play duration - Continuity across missions - Programming-based gameplay - Story-driven narrative	- Puzzle/Adventure genre - Short to medium play duration - Continuity across levels - Code editor integration - Quest-driven narrative	- Puzzle genre - Short play duration - Independent puzzles - Assembly language metaphor - Minimal narrative	- Adventure genre - Long play duration - Continuity across levels - Puzzle-based - Dialogue-driven narrative
Goal	Players program robot units using CBOT, a C++ like language to solve complex problems and progress through missions in a futuristic world.	Players learn programming by writing code to control characters and solve challenges in a fantasy setting.	Automate repetitive office tasks using a simple instruction set to solve puzzles based on assembly language principles.	Players learn and reinforce C++ fundamentals by solving puzzles, completing quests, and interacting with NPCs.
Game Dimension	3D	2D	2D	2D
Game Engine	Colobot Engine (Custom)	Custom In-House Browser-Based Engine	Tomorrow Corporation Custom Engine	Godot Engine v4.4
Features	- Single-player/Multiplayer - Robotic programming interface - Mission-based progression - Storyline-driven objectives - Teamwork mechanics (in multiplayer)	- Single-player/Multiplayer - Real-time code editor - Quest-driven progression - Level-based challenges - Collaborative features (in multiplayer)	- Single-player - Drag-and-drop command interface - Optimization challenges - Step-by-step debugging - Assembly language metaphor	- Single-player - Narrative-driven storyline - Mini-code editors for simple syntax practice - Progression system based on C++ topics - Visual storytelling elements
Accessibility	Available for download on Windows, macOS, Linux, and Android.	Browser-based, accessible via any device with internet access.	Available on PC, Mac, Linux, iOS, Android, and Nintendo Switch.	Deployable on Windows and Web (HTML5), ensuring accessibility across devices.
Suitability	Students and hobbyists interested in robotics and programming.	Learners of all ages who want to learn programming in a fun way.	Beginner to intermediate programmers interested in low-level computing concepts.	First-year diploma Computer Science students struggling with traditional teaching methods.
Methodology Used	- Programming Challenges - Mission-Based Objectives - Strategic Planning - Robotic Simulation	- Interactive Coding Challenges - Immediate Feedback - Quest-Driven Learning - Collaborative Play	- Visual Programming - Assembly Language Metaphor - Puzzle Solving - Optimization Challenges	- Puzzle-Based Learning - Level-Based Progression - Question & Feedback System - Narrative Integration - Visual Learning Aids
Evaluation Model	Kearney and Pivec's Educational Game Framework	Serious Game Constructivist Framework	Constructivist Learning Framework	MEEGA+ (Multidimensional Evaluation of Educational Games Plus)

2.7 Highlight the Chosen Techniques & Features with Justification

The design of this proposed project is based on research-backed Game-Based Learning (GBL) techniques and features that have been shown to enhance student engagement, motivation and learning outcomes, particularly in introductory programming education (Qian & Clark, 2019; van Gaalen et al., 2021). Based on findings from related educational games such as Cobot: Gold Edition, CodeCombat and Human Resource Machine, one core technique and three key features were selected for implementation:

1) Chosen Techniques

- Narrative and Storytelling

  Justification: Research shows that narrative elements increase emotional investment, contextual understanding, and memory recall in students (Kucher, 2021). In programming education, stories help frame abstract concepts within relatable scenarios, improving motivation and long-term interest (Cheong et al., 2020).

  Relevance: A strong storyline will immerse players in a meaningful world where solving programming challenges is essential to progress. This approach builds on the success of *Colobot: Gold Edition*, which uses a science fiction narrative to teach C++ inspired logic (Alves & Letouze, 2018).

Figure 2.19 Examples of games using own unique system and cutscene to deliver the storylines.

By this method as shown in figure 2.19, highlighting visual and dialogue can enhance the storytelling while providing engaging narratives experiences to students.

2) Chosen Features

- Narrative Driven Gameplay

  Justification: Emotional engagement is a powerful driver of learning. Games that integrate storylines see higher retention and motivation compared to mechanics-only designs (Kucher, 2021). Colobot: Gold Edition demonstrates that combining storytelling with programming tasks increases student interest and self-directed exploration (Heithausen, 2020).

  Relevance: The narrative will provide context and purpose to programming tasks, helping students understand the importance and practical application of what they're learning.

Figure 2.20 Examples of games using dialogue for NPC to progress the story.

Figure 2.20 shows characters interactions help a lot in guiding and engaging the players especially if being directly integrated to the gameplay.

Figure 2.21 Examples of direct narration to lead players gameplay direction

When effectively being combined with storytelling, direct narration delivers seamless gameplay mechanics in guiding players while maintaining immersion as shown in figure 2.21.

- 2D Top-Down Adventure Game Style

  Justification: The 2D top-down adventure genre supports structured exploration and puzzle-solving while maintaining visual simplicity, making it ideal for educational purposes (Prensky, 2001). It allows for navigation, clear feedback and manageable complexity is important factors when teaching beginners.

  Relevance: This genre aligns well with the goal of integrating narrative, challenge, and progression into a cohesive experience tailored for first-year diploma Computer Science students.

Figure 2.22 Examples of games using 2D top-down pixel art style genre

Figure 2.22 demonstrate visual simplicity from this game genre style is effective for structured gameplay, narrative depth and educational game designed for beginner leaners.

Figure 2.23 Examples of games using the freedom of art style creation.

Figure 2.23 shows by having the right art style direction easily transform the game concept into captivating and memorable visual experience, this will also support game's mechanics, narratives and theme goals.

- Godot 4 Game Engine

  Justification: Godot Engine offers a lightweight, open-source platform suitable for creating accessible educational games. Its ease of use, cross-platform support, and flexibility make it ideal for projects targeting educational institutions with limited resources (Dobroskok et al., 2022).

  Relevance: Using Godot enables rapid development and modifiability, supporting future enhancements or adaptations by educators. Figure 2.24, 2.25 and 2.26 shows what information and capabilities of Godot Engine.

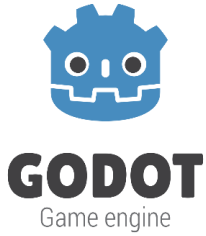


Figure 2.24 Godot 4 Game Engine Logo

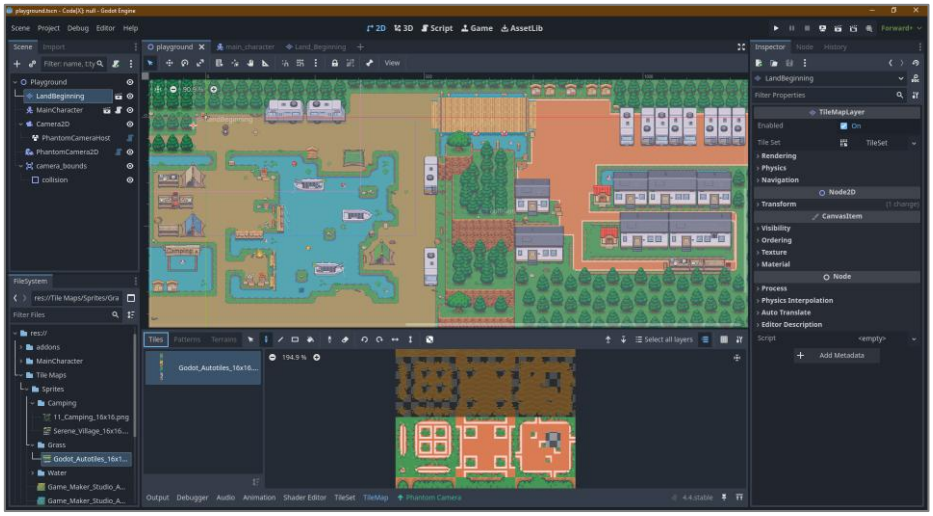


Figure 2.25 Godot 4 Game Engine Scene Editor Interface

In figure 2.25, the Godot 4 Engine Scene editor is a visual tool used for designing and assembling game scenes, allowing placing nodes in a hierarchical structure, configure their properties, and interactively preview the scene layout and behavior based on developer's creativity.

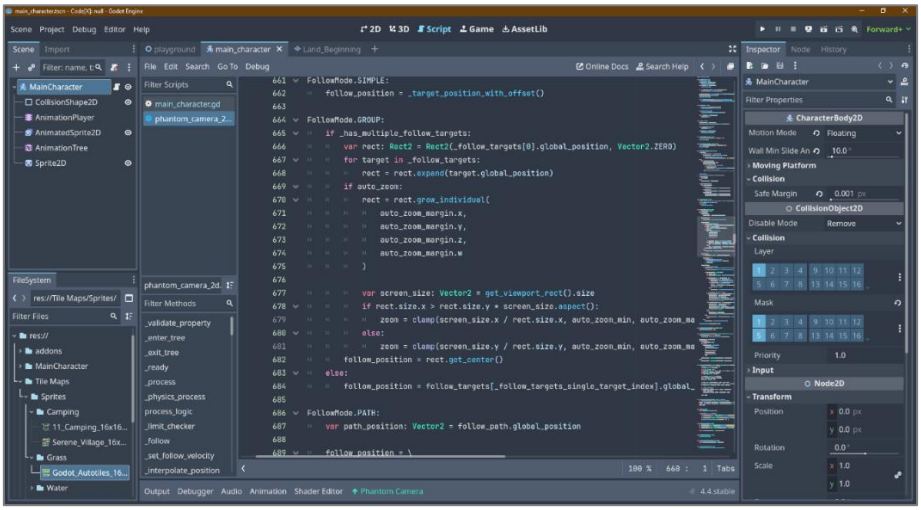


Figure 2.26 Godot 4 Game Engine Code Editor Interface.

In figure 2.26, this is where all mechanics like player movement, collision, animation and the core of game elements implementation will be done inside of here. Godot uses its own programming language which is the GDScript, a high-level, object-oriented, imperative and gradually typed programming language built specially for Godot Engine.

2.8 Summary

This literature review focuses the need for more engaging methods in Computer Science education, particularly for first-year diploma students who often struggle with abstract programming concepts. Traditional teaching methods are typically passive and struggle to sustain interest in topics like algorithms and data structures.

Game-Based Learning (GBL) offers a compelling alternative by incorporating elements like progression systems, challenges, and storytelling to promote active learning and motivation. Educational games such as Colobot: Gold Edition, CodeCombat and Human Resource Machine demonstrate the potential of GBL, though few effectively combine all key elements.

To fill this gap, the proposed project introduces a narrative-driven 2D top-down adventure game built with Godot 4 Game Engine. It uses question-based puzzles, mastery-driven progression, and narrative storytelling to support self-paced learning of introductory C++ programming.

By blending storytelling, gameplay and interactivity, the proposed project aims to improve the accessibility and effectiveness of programming education for the first-year diploma students in Computer Science education.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter presents the methodology adopted in the design and development of proposed project. The methodology follows a structured Game Development Life Cycle (GDLC) approach, ensuring that each phase contributes directly to achieving the project's core objectives.

Each phase of the GDLC is aligned with educational goals, ensuring that gameplay mechanics such as puzzles, progression systems and storytelling are not only entertaining but also pedagogically sound and technically feasible. This section provides a concise overview of how the game was conceptualized, designed, implemented, tested and prepared for release, while emphasizing the use of the Godot Engine v4.4 as the development platform. The subsequent sections will elaborate on the specific phases of the GDLC, system architecture and technical requirements.

3.2 Project Methodology

The development of followed a Game Development Life Cycle (GDLC) framework to ensure alignment with pedagogical goals and technical feasibility. This structured approach enabled systematic progression from conceptualization to deployment, ensuring the game meets the needs of first-year Computer Science students. Figure 3.1 shows the project methodology of the game development life cycle.

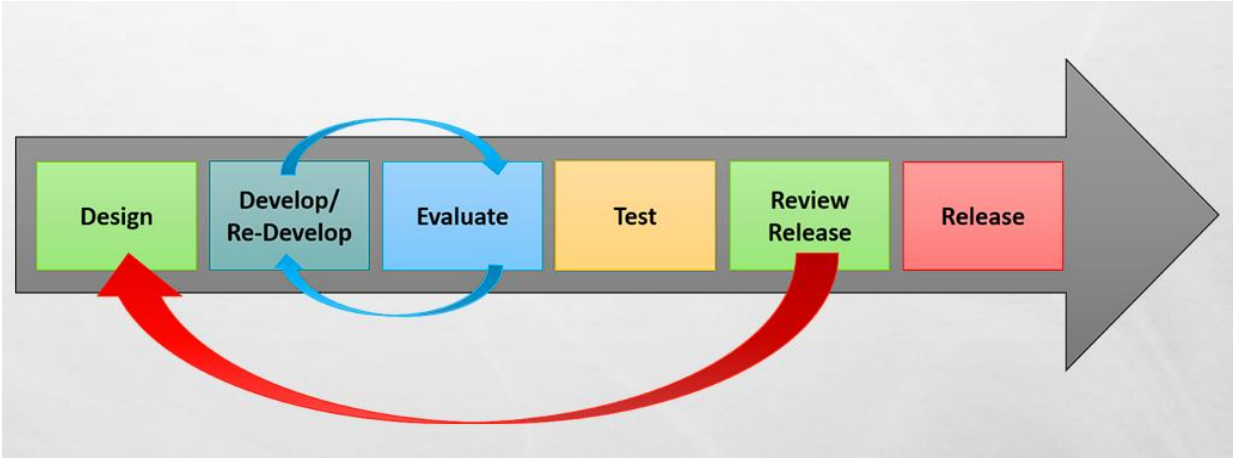


Figure 3.1 Game Development Life Cycle

The project was divided into six iterative phases, as outlined in Table 3.1. Each phase was tailored to address specific project objectives:

Table 3.1 Project Methodology (GDLC)

Phases	Activities	Technique/Software	Deliverable
Design	• Create narrative-driven storyline • Map C++ concepts to gameplay puzzles • Plan UI/UX elements	• Canva and draw.io	• Narrative design document • Scene/dialogue layout and UI/UX diagrams
Develop / Re-Develop	• Build game scenes in Godot Engine • Script puzzles using GDScript • Create assets (sprites, sounds)	• Godot Engine v4.4 • GDScript • Aseprite / Piskel • Audacity / Bfxr • Git	• Functional game prototype

	• Test basic functionality		
Evaluate	• Review puzzle accuracy • Validate with instructors • Cross-check against curriculum	• Excel / Google Sheets • Feedback forms	• Revised learning content
Test	• Conduct alpha testing • Collect player feedback • Identify bugs • Measure cognitive load	• Google Forms • Trello / Notion • Playtesting sessions	• Player feedback summary
Review Release	• Finalize graphics and audio • Optimize performance • Prepare export templates • Write instructions manual.	• Godot Export • Performance profiling tools • Documentation editors	• Stable build for deployment
Release	• Publish on Itch.io • Distribute to target users • Gather usage data • Monitor feedback	• Web hosting (Itch.io Page) • Google Forms	• Deployed finished game version

3.2.1 Design

This phase focused on identifying effective Game-Based Learning (GBL) methods. By integrating narrative storytelling and puzzle-based challenges, the project aimed to enhance student engagement and comprehension. Drawing from recent research on GBL's impact on motivation (Kroustalli & Xinogalos, 2021), the narrative was designed to contextualize C++ concepts (e.g., variables, loops) within a cohesive storyline. To achieve this, several key design artifacts were produced, including storyboard drafts, concept art, and dialogue scripts, ensuring that all gameplay mechanics align with pedagogical goals to foster a structured yet interactive learning environment.

A. Storyboard and Narrative Flow

Storyboarding was used to visualize the sequence of events, player interactions, and the integration of learning puzzles within the narrative. A key storyboard sequence, for example, outlines the player's quest, where they encounter a broken "transformer" and must use their knowledge of C++ variables to repair it. This process ensures the narrative pacing and puzzle difficulty are balanced.

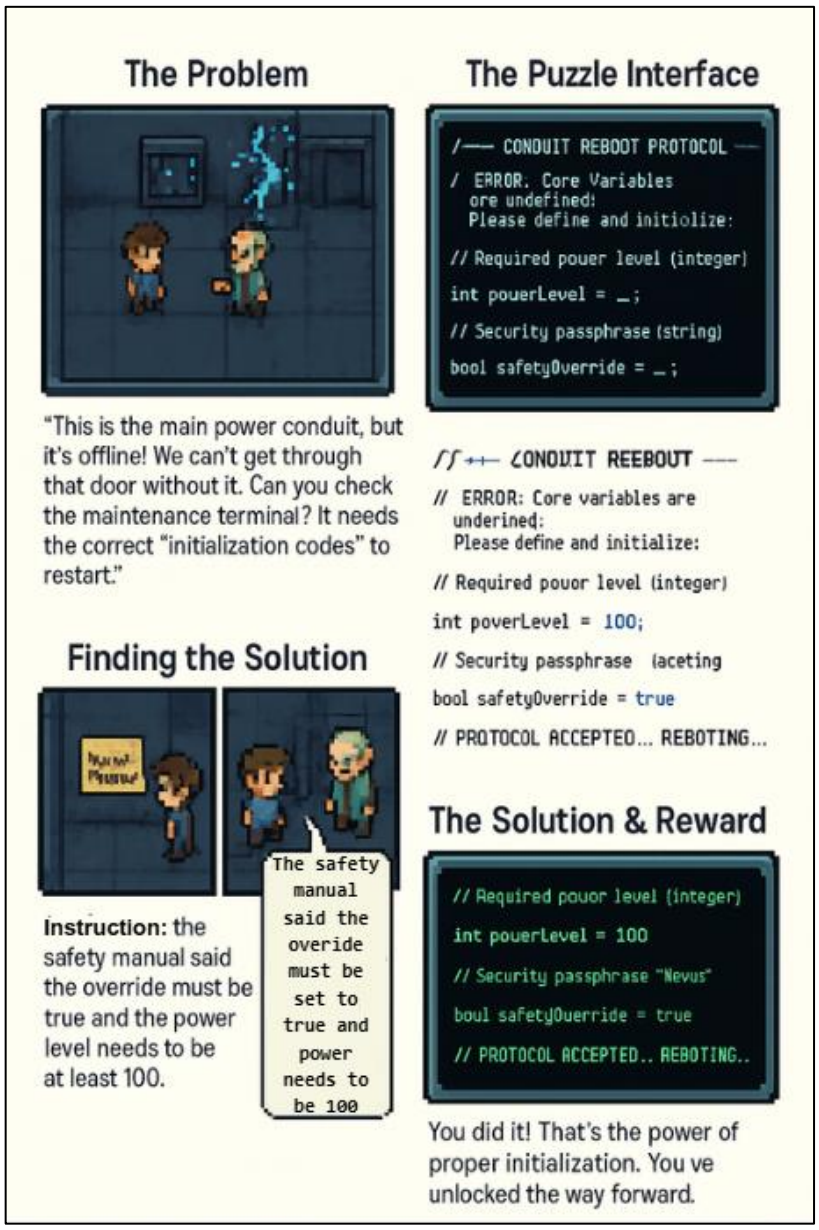


Figure 3.2 Storyboard Draft for the "Conduit's Variable" Quest

From the figure 3.2, this storyboard sequence visualizes the "Conduit's Variable" quest, which is designed to teach the fundamental C++ concepts of variable declaration and initialization. It depicts the narrative flow where the player is presented with a problem, gathers information from the environment and NPCs, and applies their knowledge to solve a puzzle. This process directly integrates learning (int, string, bool data types) with gameplay, providing immediate feedback and in-game progression as a reward.

B. Concept Art and World Building

To establish the visual identity and atmosphere of "Code[X]:Nexus", concept art was developed for the main character and the game world. The 2D top-down pixel art style was chosen for its clarity and low-performance overhead, ensuring accessibility. The character design aims to be relatable for students, while the world design combines elements of a futuristic tech environment with classical adventure game tropes.

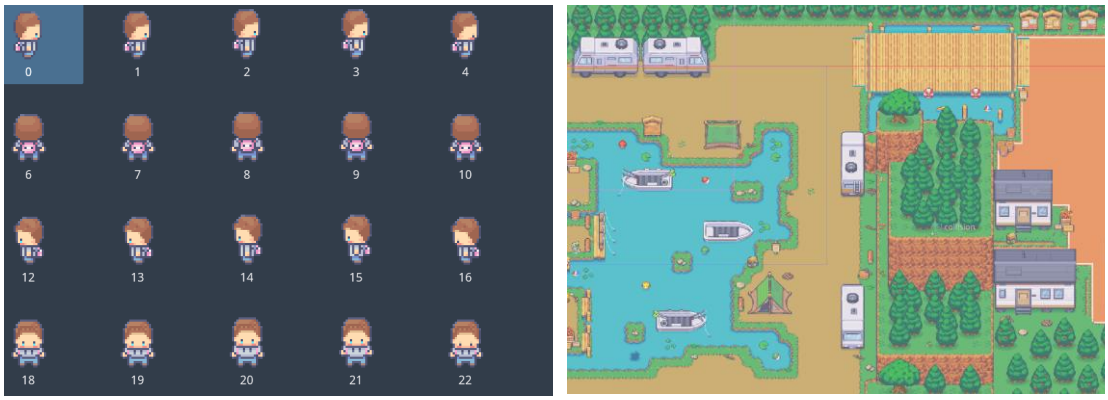


Figure 3.3 Concept Art for the Player Character and a Game World Area

Figure 3.3 shows the player character sprite with 23 animation frames (left) and a concept mockup of "Vector Valley," the starting game area (right).

C. Dialogue and Narrative Integration

Dialogue is a critical component for delivering the narrative and framing the C++ challenges. Scripts were written to be engaging while subtly embedding educational content. The dialogue not only guides the player through quests but also serves as a hint system and provides contextual feedback for the puzzles. Below (Table 3.2) is a sample interaction where an NPC introduces the concept of for loops.

Table 3.2 Concept of the designed project quest's dialogues.

Character	Dialogue
Professor Ada	"Greetings, young lads. This data stream is fragmented. We need to process every packet, one by one."
Player	"How can I do that?"
Professor Ada	"Think in cycles. A for loop would be perfect. It can initialize a counter, set a condition to stop, and increment through each packet. Can you build the loop structure to fix it?"

3.2.2 Develop / Re-Develop

This phase addressed developing a 2D top-down adventure game using the Godot Engine v4.4. Leveraging Godot's node-based architecture and GDScript, the game's core mechanics includes character movement, quest triggers and puzzle logic were implemented. Assets like sprites and audio were created using tools like Aseprite and Audacity, ensuring the game met technical specifications (e.g., low system requirements) for accessibility in educational settings (Dobroskok et al., 2022). Iterative development allowed for refinement of gameplay elements, such as interactive dialogue systems and level transitions, to align with the project's narrative-driven vision.

3.2.3 Evaluate

This phase aimed to assess the game's effectiveness. It involved cross-checking puzzle accuracy against C++ curriculum standards and validating content with instructors. For example, puzzles on variables and conditionals were reviewed to ensure they accurately reflected foundational programming concepts, ensuring pedagogical alignment (Holly et al., 2024). Feedback from early testers was used to refine learning modules, directly supporting the objective of evaluating the game's instructional value (Soleymani et al., 2025).

A. Evaluation Participants and Activities

The evaluation involved first-year Computer Science students who represented the target end-users of the game. These participants were selected based on their current enrollment in introductory C++ programming courses, ensuring they possessed the appropriate knowledge level to assess content relevance. During evaluation sessions, participants engaged in guided walkthroughs of the game prototype, critically reviewing elements such as in-game puzzles, dialogue, and instructional text to assess their clarity, difficulty appropriateness, and alignment with their learning experience. The project supervisor oversaw the evaluation process, providing guidance and conducting a final

review to confirm the validity and reliability of the findings before finalizing the assessment outcomes.

B. Evaluation Criteria

The game's content was evaluated using criteria grounded in pedagogical best practices and the project's objectives, adapted from models like MEEGA+ (Multidimensional Evaluation of Educational Games Plus). Student evaluators assessed four key areas: Content Clarity (whether C++ concepts were explained understandably), Difficulty Appropriateness (whether puzzles matched their skill level), Interface Usability (ease of navigation and interaction), and Engagement Potential (motivation to continue playing). Feedback from this phase informed refinements to learning modules and puzzle designs, ensuring the game was both educationally effective and engaging before the larger-scale alpha testing began.

3.2.4 Test

This phase involves alpha testing to evaluate the game's usability, engagement, and preliminary learning outcomes. A small group of 31 Computer Science diploma students will participate in structured playtesting sessions. The evaluation uses a mixed-methods approach: direct observation of player behavior, a post-session survey (via Google Forms) with scales and open-ended questions to assess usability, engagement, and comprehension, and short informal interviews for qualitative insights. Feedback and identified bugs are logged and prioritized for iteration. This combination of observation, survey, and interview serves as the primary mechanism to evaluate engagement.

3.2.5 Review & Release Preparation

This phase covers the final review and preparation activities required to create a polished and stable build of the game for PC deployment. Before creating the final build, a comprehensive internal review is conducted, which includes a code review for efficiency, a final asset review for all graphics and audio, a functional review to test all game mechanics, and a documentation review for the user guide. For release, the game is prepared for the PC (Windows) platform. This involves using Godot's export tools to generate a standalone executable file (.exe) and packaging all necessary assets into a single .zip file suitable for distribution on platforms like Itch.io. A key activity during this phase is performance optimization; performance profiling tools are used to reduce load times and ensure the game is accessible on low-end devices, which is critical for classroom use. As the release is focused on PC, requirements for other platforms like the Google Play Store are not applicable. The detailed minimum and recommended hardware specifications required to run the game are formally outlined later in this document in Section 3.5.

3.2.6 Release

This phase enabled gathering real-world data on effectiveness. By publishing the game on platforms like Itch.io Page and distributing it to students, the project collected initial user feedback to inform future iterations. Metrics like playtime, completion rates and post-game surveys provided insights into the game's impact on student motivation and learning outcomes (Kroustalli & Xinogalos, 2021).

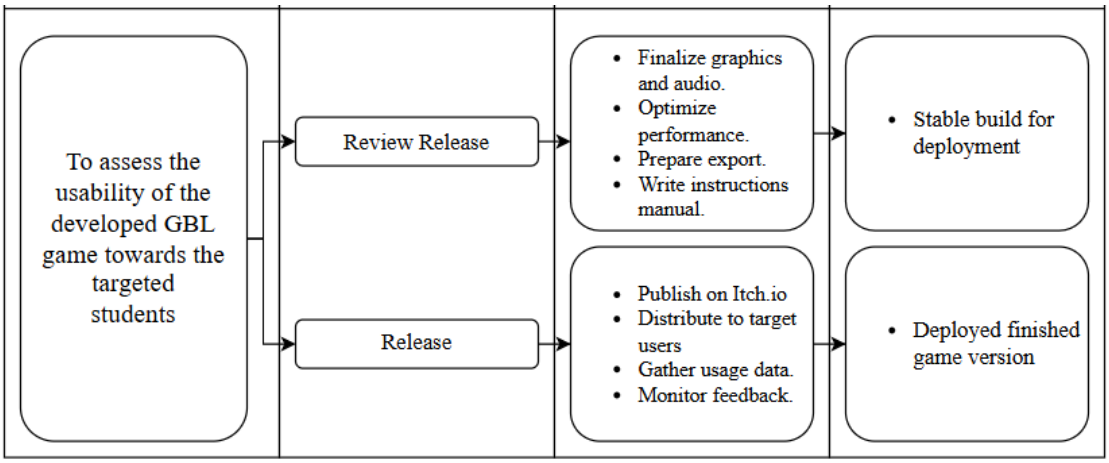
Together, the GDLC phases created a systematic development process that directly addressed each project objective, from conceptualizing GBL methods to deploying a functional, engaging educational tool.

3.3 Summary of Project Methodology

This section presents a summary of how each phase in the Game Development Life Cycle (GDLC) aligns with the specific objectives of the proposed project, ensuring that every stage of development directly supports the educational and technical goals outlined in Chapter 1 as shown in the table 3.3 for the project methodology summary.

Table 3.3 Summary of Project Methodology

Objective	Phases	Task	Deliverables
To identify an effective game-based learning (GBL) technqiues in programming education in enhance student motivation	Design	• Create narrative-driven storyline. • Map C++ concepts to gameplay puzzles. • Plan UI/UX elements.	• Narrative design document • Scene layout and UI/UX diagrams
To develop a 2D top-down style learning adventure game, in learning programming technqiues that includes GBL elements using Godot Engine v4.4	Develop / Re-Develop	• Create narrative-driven storyline. • Map C++ concepts to gameplay puzzles. • Plan UI/UX elements.	• Create narrative-driven storyline. • Map C++ concepts to gameplay puzzles. • Plan UI/UX elements.
	Evaluate	• Review puzzle accuracy. • Validate with instructors. • Cross-check against curriculum.	• Revised learning content
	Test	• Conduct testing. • Collect player feedback. • Identify bugs • Measure cognitive load.	• Player feedback summary



As the summary illustrates, the project methodology provides a clear and structured workflow. Each project objective is systematically mapped to distinct development phases (from Design to Release) and tied to concrete tasks and deliverables. This methodical alignment ensures that every development effort is purposefully directed toward solving the core challenge of student disengagement. By following this approach, the project ensures the final product is not merely a functional game, but a cohesive, narrative-driven educational experience designed to improve learning outcomes for introductory C++ students.

3.4 System Architecture

The system architecture of proposed project was designed to ensure a modular, scalable and pedagogically effective structure that supports both gameplay mechanics and educational content delivery. The game will be developed using the Godot Engine v4.4, leveraging its node-based scene system and lightweight performance for accessibility on low-end systems targeting deployment on Windows PC and HTML5 Web Browser.

The game follows a Node-Based Scene Architecture, a core feature of the Godot Engine. In this style, each game component (e.g., player, NPC, puzzle) is represented as a node within a hierarchical scene graph. Nodes encapsulate specific behaviors (e.g., movement, dialogue) and can be reused across levels, promoting code reusability and scalability (Dobroskok et al., 2022). This is the native architecture style used in Godot, where each scene contains a hierarchy of nodes that handle specific roles such as player movement, dialogue display, puzzle logic, UI elements, and progress tracking. Each scene can be treated as a self-contained module, which aligns with best practices in software design and supports reusability and scalability.

This architecture style offers several key functionalities: modular scenes can be reused across different levels, making it easy to add new puzzles or narrative segments without disrupting existing code. Additionally, there is a clear separation between gameplay, UI, and logic layers, which enhances maintainability and scalability. The design is also lightweight and efficient, ensuring compatibility with low-end systems.

A. User Interface Layer

This layer encompasses all the visual elements the player interacts with, including menus, the Head-Up Display (HUD), dialogue boxes, and the puzzle interface. It is built using Godot's native Control nodes and CanvasItem system. The primary goal of this layer is to ensure clarity and accessibility, promoting an intuitive interaction model that is easy for first-year students to grasp and use effectively.

B. Game Logic Layer

The Game Logic Layer is the core engine of the gameplay experience, responsible for managing player input, character movement, the triggering of quests, and transitions between levels. It utilizes GDScript, a Python-like language chosen for its simplicity and readability, to script all gameplay events. This layer also includes state machines to govern NPC behavior and validate puzzle solutions, ensuring the game world is dynamic and responsive to player actions.

C. Learning Content Layer

This layer directly integrates the educational objectives into the game by containing embedded C++ programming puzzles that are mapped to specific curriculum topics. Each puzzle is designed to represent a fundamental concept such as variables, loops, conditionals, or functions, thereby reinforcing theoretical knowledge through direct, practical application. The design is inspired by games like "Human Resource Machine," where players solve algorithmic problems through visual interaction to foster computational thinking.

D. Narrative Integration Layer

To enhance engagement and provide a contextual framework for learning, the Narrative Integration Layer weaves the educational content into a compelling story. It uses story-driven missions and branching dialogue trees to guide the player's progression through the game. This approach is inspired by successful educational games that leverage narrative to motivate learners and create a cohesive, meaningful experience.

E. Data Management Layer

This layer is responsible for the persistence of player information, which includes tracking progress, saving game states, and logging performance data. This functionality enables personalized learning paths and allows for effective progress monitoring. It uses a simple file-based storage system, which eliminates the need

for an external database, thereby simplifying deployment and reducing system overhead. This also provides players with the convenience of resuming gameplay from their last completed level.

High-Level System Architecture Diagram (Descriptive)

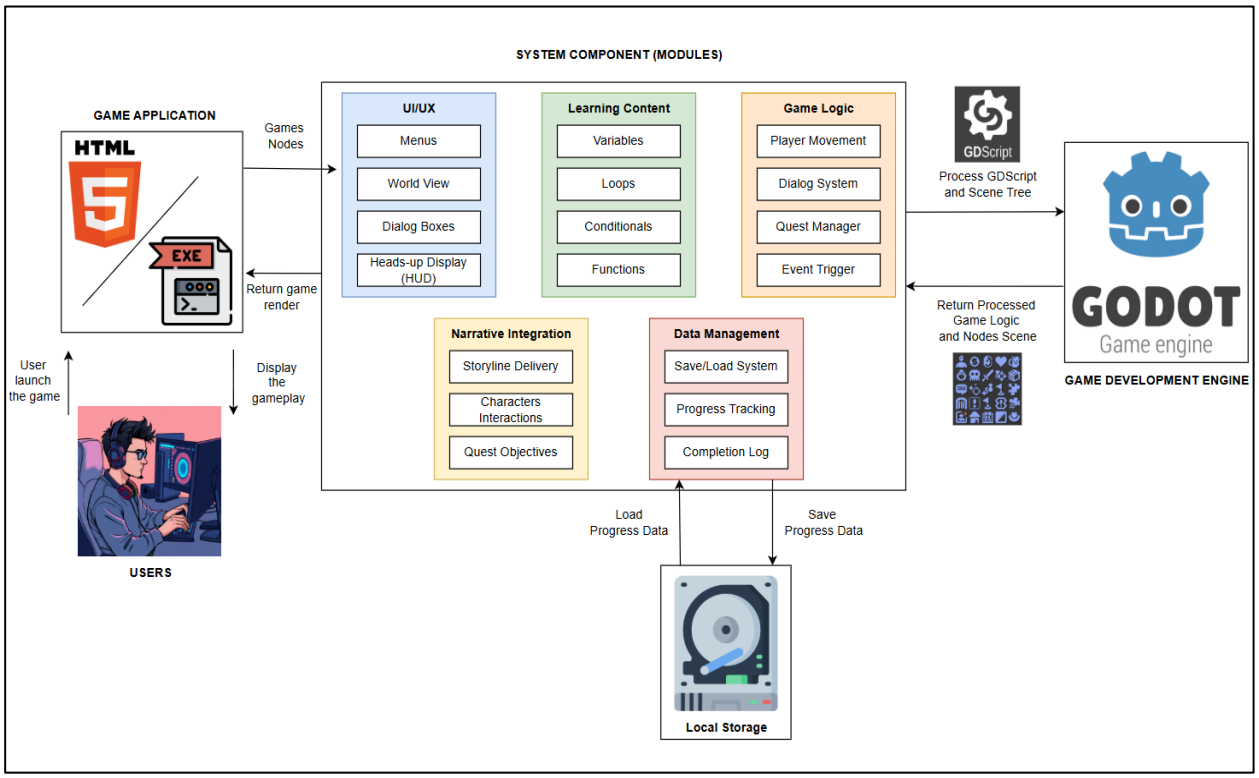


Figure 3.4 System Architecture Design of Proposed Project

Based on Figure 3.4, the System Architecture Design for "Code[X]: Nexus" illustrates a modular structure centered around the Godot Game Engine. Users launch the Game Application, which processes game nodes and returns rendered gameplay. The core System Components are divided into: UI/UX, Learning Content, Game Logic, Narrative Integration, and Data Management. All game data is persistently stored and retrieved from Local Storage, ensuring player progress is maintained. The Godot Engine processes GDScript and scene trees, returning processed game logic and scenes to the application for display, thus orchestrating the entire game experience.

Flowchart Design Diagram

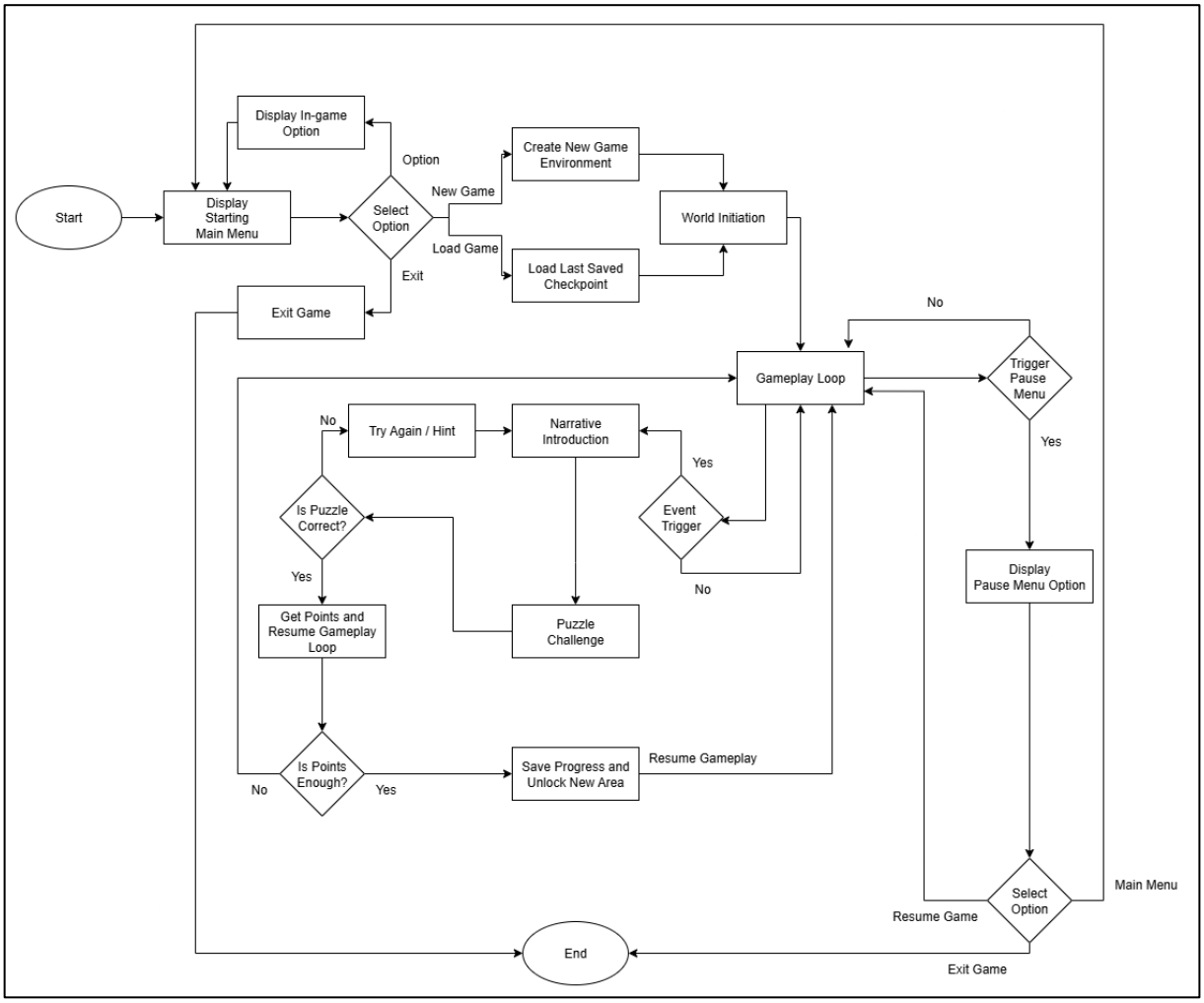


Figure 3.5 Flowchart diagram for the proposed project

Based on Figure 3.5, the operational flow of "Code[X]: Nexus" begins at the start state, presenting the player with a main menu to initiate a new game, load game, access in-game options or exit game. The core gameplay unfolds within the gameplay loop, which can be paused to access menu options for resuming, returning to the main menu or exiting. Within the loop, event triggers lead to narrative introductions and puzzle challenges. Successful puzzle completion awards player with points which will leads in unlocking new area if sufficient points are accumulated. Thus, the game process concludes at the End state if exiting the game.

Use Case Diagram Design Diagram

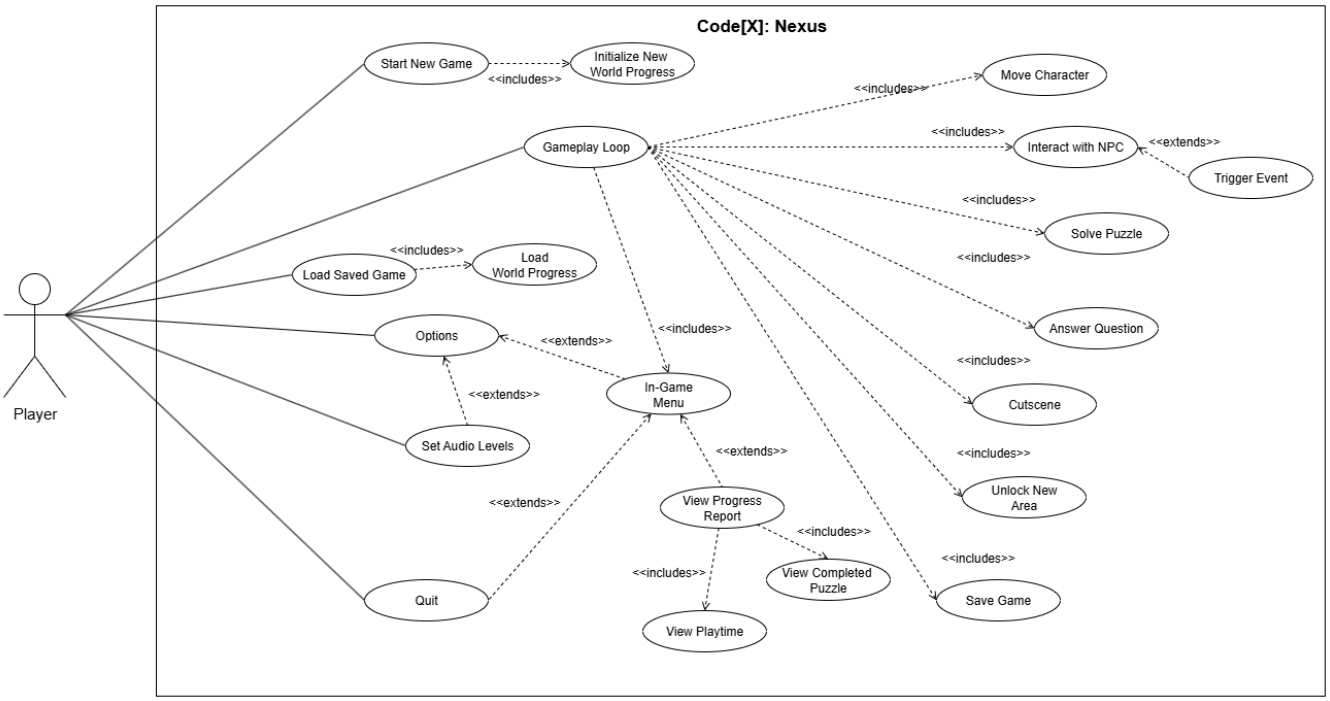


Figure 3.6 Use case diagram of the proposed project

Based on the Figure 3.6, the use case diagram illustrates how the Player interacts with the game system to achieve various goals. The Player can manage their game experience by entering the main menu to start a new game, load a saved one, adjust options like audio levels, or quit. Once in the Gameplay Loop, the Player actively engages with the game world by navigating environments, interacting with NPCs (which may involve dialogue or triggering story events), and experiencing the narrative. Central to the game's purpose, the Player also solves programming challenges, providing code solutions and receiving immediate feedback. As they progress, the Player can unlock new content and track their achievements by viewing their overall progress, including completed challenges and playtime. Essentially, the diagram outlines all the key functionalities the game offers to its sole user, the Player.

3.5 Hardware and Software Requirements

To ensure broad accessibility and ease of use, "Code[X]:Nexus" was developed with minimal system requirements in mind, making it suitable for deployment in educational environments where students may be using low-end devices or shared computing resources (Tan et al., 2021). The game is designed to run efficiently on standard classroom computers, personal laptops or tablets. This section outlines both the software tools used during development and the minimum hardware specifications required for end-users to play the game. Table 3.4 and 3.5 shows the required software and hardware requirements needed.

Software Requirements:

Table 3.4 Software Requirements Details

Component	**Description**
Game Engine	Built with Godot Engine v4.4, an open-source engine for rapid prototyping and cross-platform deployment.
Programming Language	Gameplay logic was implemented in GDScript for its simplicity and readability.
User Interface Design	UI elements such as dialogue boxes, puzzle interfaces and HUD components were created using Godot's built-in node system.
Asset Creation Tools	• Piskel and Aseprite for creating pixel art graphics. • Bfxr and Audacity for sound effects and music. • Trello and Notion for task tracking and documentation.
Version Control	Git and GitHub were used for version control and collaboration during development.
Export Targets	The final build was exported for two platforms: • Windows PC • HTML5 Web Browser

Hardware Requirements:

Table 3.5 Hardware Requirements Details

Requirements	Minimum Specification	Recommended Specification
Operating System	Windows 7 SP1 or higher / Web browser (Chrome, Firefox, Edge)	Windows 10 (64-bit) or higher / Modern web browser (latest versions)
Processor	Intel Core i3-2100 or equivalent (AMD A6-Series or equivalent)	Intel Core i5-3570 or better (AMD A8-7410)
RAM	1 GB DDR3 SDRAM	2 GB DDR3 SDRAM
Graphics	Integrated graphics card (e.g., Intel HD Graphics 2000 or AMD Radeon HD 6310)	Dedicated GPU not required; mid-range integrated GPU (e.g., Intel UHD Graphics 630/ Radeon HD 7660G)
Storage	200 MB available space on the primary drive (HDD or SSD)	500 MB available space on the primary drive (SSD preferred)
Input Device	Keyboard and mouse (PS/2 or USB)	Keyboard, mouse or gamepad support (USB)

The minimal requirements reflect the project's focus on accessibility for first-year students, many of whom may use low-end devices in classroom settings (Tan et al., 2021). By supporting both Windows and HTML5, the game can be deployed in labs, libraries, or at home, aligning with the objective of creating an "anytime, anywhere" learning tool (Cheong et al., 2020).

3.6 Conclusion

This chapter outlined the methodology used in the development of “Code[X]:Nexus”, a narrative-driven 2D top-down adventure game designed to teach introductory-level C++ programming to first-year diploma students. The development process followed a structured Game Development Life Cycle (GDLC), ensuring that each stage starting from initial design to final release was aligned with both educational goals and technical feasibility.

Each phase of the GDLC contributed directly to the project's core objectives, starting with the Design phase, which established a strong foundation for integrating narrative and learning content. The subsequent Development phase implemented these ideas using the Godot Engine v4.4, leveraging its accessibility for educational purposes. Rigorous Evaluation and Testing phases then ensured the learning modules were accurate, engaging, and appropriately challenging. Finally, the Release phase made the game available to its target audience, enabling valuable feedback collection for future improvements.

By following this systematic approach, the proposed project was developed as an accessible, engaging and educationally meaningful tool that bridges the gap between traditional instruction and interactive learning. The modular system architecture and minimal hardware requirements ensure that the game can be used in a variety of educational environments, including classrooms with limited technological resources.

CHAPTER 4

RESULT AND DISCUSSION

4.1 Introduction

The purpose of this chapter is to outline the results and outcomes from the development process for Code[X]: Nexus, which is a top-down 2D adventure game that uses an algorithmically-based and narrative approach to introduce users to introductory C++. Results were grouped by three main categories: the implementation of the games logic and algorithms, the design of the game environment and UI, the development of the character models, and the technical aspects of deploying the system to a user's computer. This chapter also outlines the build and optimizations necessary for the system to be compatible with both PC and Mac platforms. Finally, the chapter will detail the quantitative and qualitative data collected during user testing. It will evaluate the degree to which the GBL methods selected for this project meet the objectives of the project as stated in Chapter 1, and confirm the success of developing a viable learning tool utilizing the Godot Engine.

4.2 Educational Game Algorithm and Logic Implementation

Code[X]: Nexus is based on a modular, node-based architecture in GDScript which has been defined in chapter 3 as well as the system architecture. This section will detail the specific algorithm that have been developed for the handling of the educational puzzle and the progress of the player.

4.2.1 State Management Algorithm (Quest & Progress Tracking)

To ensure a continuous narrative experience across different game scenes (World 1: Dark Woods, World 2: Zen Garden, World 3: Hollowed Core, World 4: Horizon's End), a persistent state management system was implemented. This was achieved using a custom Autoload singleton script, QuestManager.gd as shown in Figure 4.1

```
74  # 2. We store the default state in a constar
75  const DEFAULT_QUEST_DATA = {
76      # World_Start scene
77      "met_aliff": false,
78      "met_jackal": false,
79      "met_adli": false,
80      "met_maya": false,
81      "met_alya": false,
82      "got_nexus": false,
83      "intro_monologue_played": false,
84      "map_monologue_played": false,
85
86      # World_1 Scene
87      "world_1_monologue_played": false,
88      "met_zafran": false,
89      "solved_darkness_puzzle": false,
90      "met_bukh": false,
91      "solved_bukh_puzzle": false,
92      "met_lyra": false,
93      "solved_lyra_puzzle": false,
```

```
# It checks only the objectives for a specific world
func are_world_objectives_complete(world_name: String) -> bo
    var objectives_to_check = []

    match world_name:
        "world_start":
            objectives_to_check = WORLD_START_OBJECTIVES
        "world_1":
            objectives_to_check = WORLD_1_OBJECTIVES
        "world_2":
            objectives_to_check = WORLD_2_OBJECTIVES
        "world_3":
            objectives_to_check = WORLD_3_OBJECTIVES
        _:
            printerr("QuestManager: Unknown world name '", w
            return false
```

Figure 4.1: QuestManager Dictionary handling boolean flags for game state.

The algorithm uses a dictionary data structure to keep track of Boolean flags for key events that occur within the game, i.e. the first time you meet an NPC (met_alyra = True), the first time you solve a particular code challenge (solved_loops = False), etc.

- **Initialization:** At game launch, the system will load a default set called DEFAULT_QUEST_DATA.

- **Persistence:** As the user advances through the game, the QuestManager continuously updates these Boolean values in real-time. This allows the game to "remember" the user's actions, whether they are entering a new scene, e.g. going through a portal, or if they are relaunching the game from the main menu.
- **Gating:** Algorithms in the interaction logic within NPC script files (i.e. Asad_NPC.gd) will evaluate the flags prior to allowing the user access to a new branch of dialogue and will ensure the user is learning content in a logical educational sequence (Variables > Loops > Conditional Statements > Functions).

4.2.2 C++ Compiler Puzzle Logic

The compiler is the primary game-based learning mechanism within the game. The compiler represents an attempt to increase players' comprehension of programming concepts by simulating a C++ compiler. The logic behind this feature was developed in the CompilerUI scene as shown in Figure 4.2.

SUCCESS

```
func _on_run_button_pressed():
    # --- 1. Disable UI immediately to prevent spamming ---
    run_button.disabled = true
    code_input.editable = false

    var player_code: String = code_input.text

    var normalized_player_code = _normalize_code(player_code)
    var normalized_solution = _normalize_code(current_solution)

    if normalized_player_code == normalized_solution:
        # --- SUCCESS ---
        correct_answer_sfx.play() # <-- PLAY SUCCESS SOUND
        hint_label.text = "Success! The code compiled and the
        system is responding!"
        npc_portrait.texture = tex_success

        # --- NEW: Set hint color to green (optional) ---
        hint_label.add_theme_color_override("font_color", Color.GR

        # (UI stays disabled)

        if current_quest_id:
            SignalManager.quest_step_completed.emit(current_quest_

        await get_tree().create_timer(2.0).timeout
        close_window()
```

FAILURE

```
else:
    # --- FAILURE ---
    wrong_answer_sfx.play() # <-- PLAY FAIL SOUND
    hint_label.add_theme_color_override("font_color", Color.RED)
    hint_label.text = "Compiler Error: 'Syntax Error'. That's
    not quite right. Check your logic and try again."
    npc_portrait.texture = tex_fail # <-- SET FAIL PORTRAIT

    # (Optional) You can print this for debugging:
    print("Player code: ", normalized_player_code)
    print("Solution code: ", normalized_solution)

    # --- 2. THIS IS THE NEW PART ---
    # Wait for 2 seconds
    await get_tree().create_timer(2.0).timeout

    # --- 3. REVERT THE UI ---
    if is_visible():
        npc_portrait.texture = tex_default # <-- REVERT TO DEFAUL

        # --- Revert hint text and color ---
        hint_label.text = current_hint
        hint_label.remove_theme_color_override("font_color")

        # Re-enable the UI for another try
        run_button.disabled = false
        code_input.editable = true
        code_input.grab_focus() # Put the cursor back in the text
```

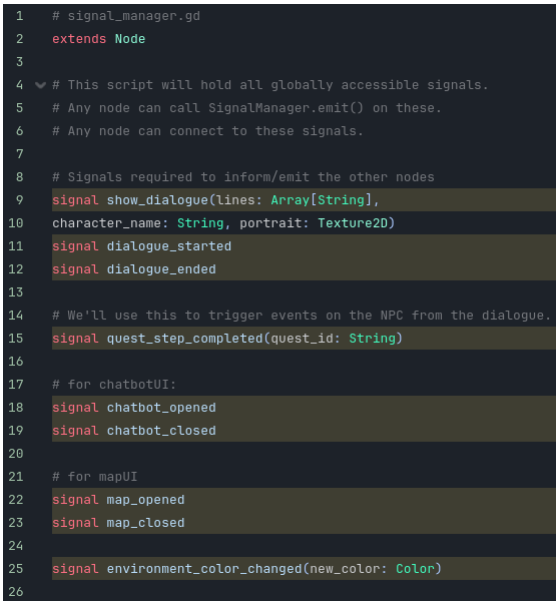
Figure 4.2: Input validation algorithm in GDScript checking for correct C++ syntax.

The algorithm for validating player input functions as follows:

1. **Input Normalization:** A To ensure that the player's inputs were being evaluated based upon logic and not formatting (for example, x = 10; would be treated identically to x=10;) a custom string processing function, _normalize_code() ,was created. It processes all user input to remove all leading or trailing whitespace, tabs, and new line characters.
2. **Logic Validation:** After normalization of the user's input, it is then compared to a predefined "Solution Key" in the NPC's data dictionary.
3. **Feedback Loop:**
    - If the user's input is deemed correct, a "Success" signal is sent and displays a green text visual success state along with a update of the QuestManager.
    - If the user's input is incorrect, a "Syntax Error" message will display, mimicking how a real IDE error log would look and encourage the player to debug their solution.

4.2.3 Event-Driven System Architecture

In order to ensure a completely decoupled and efficient code base, a SignalManager system was developed. This allows the various components of the game to be independent of one another (Player, UI, etc.) and allows them to communicate with one another via signals, thus aligning with the “Game Logic Layer” design philosophy as shown in Figure 4.3.



```gdscript
# signal_manager.gd
extends Node

# This script will hold all globally accessible signals.
# Any node can call SignalManager.emit() on these.
# Any node can connect to these signals.

# Signals required to inform/emit the other nodes
signal show_dialogue(lines: Array[String],
character_name: String, portrait: Texture2D)
signal dialogue_started
signal dialogue_ended

# We'll use this to trigger events on the NPC from the dialogue.
signal quest_step_completed(quest_id: String)

# for chatbotUI:
signal chatbot_opened
signal chatbot_closed

# for mapUI
signal map_opened
signal map_closed

signal environment_color_changed(new_color: Color)

```

Figure 4.3: SignalManager singleton centralizing global game events.

- **Global Signals:** Events such as dialogue_started, quest_step_completed, and environment_color_changed are broadcast globally.
- **Observer Pattern:** Objects like the Player script listen for these signals. For example, when dialogue_started is emitted, the Player script automatically disables movement logic (can_move = false), preventing input conflicts during cutscenes or conversations.

4.3 Game User Interface (UI) and Environment Design

Designing the visual appearance of the game to enable an immersed learning experience. The "User Interface Layer" from the System Architecture was used to help guide this design effort.

4.3.1 Scene Management and Integration

Two distinct interfaces were developed to facilitate the learning content:

- **The Compiler Interface:** A simple IDE (Integrated Development Environment) that mimics a real-world coding environment. The broken code is shown on a syntax highlighted display and a plain text area where the student can write their solution. Students become accustomed to standard coding environment interfaces when they work with the compiler interface. Compiler interface design is as shown in Figure 4.4.

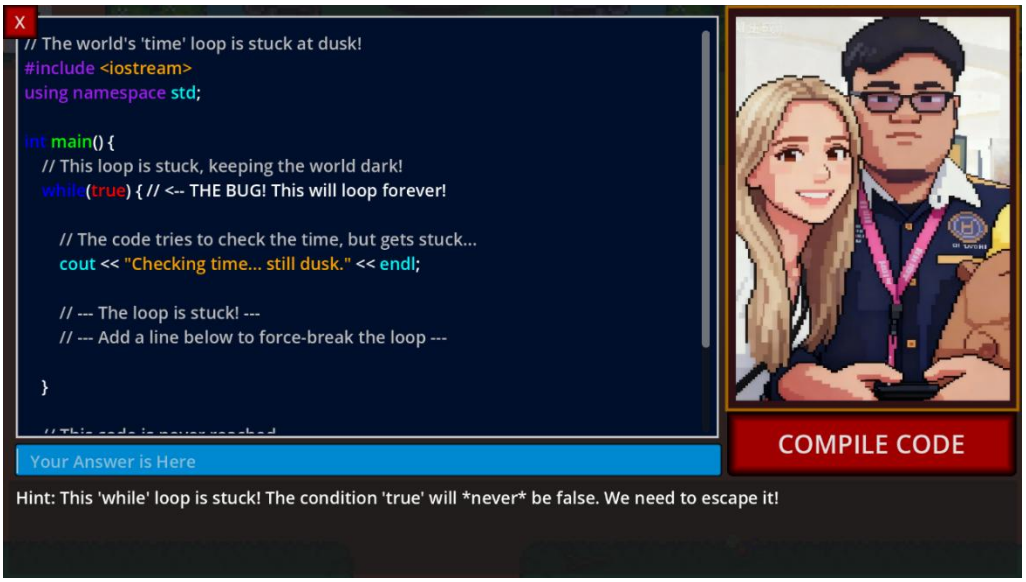


Figure 4.4: The simulated C++ Compiler Interface allowing users to debug broken code.

- **The Logic Puzzle Interface:** A choose and select based UI (Example: Word Matching) was used to show the "word puzzle" problems. The purpose of this type of user interface was to help reinforce the terms of the puzzles being learned (i.e., matching "integer" to "whole number") while minimizing the amount of cognitive load associated with entering the correct syntax. Puzzle design is as shown in Figure 4.5.

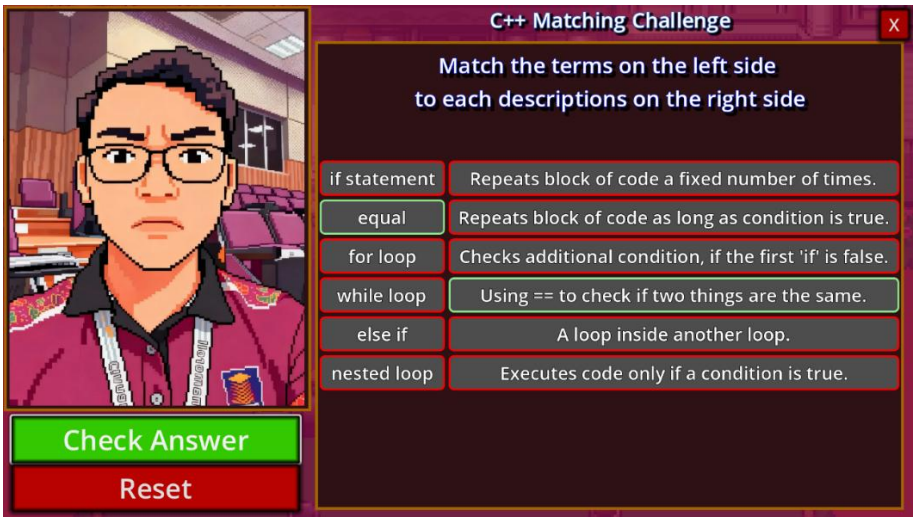


Figure 4.5: Logic Puzzle Interface for reinforcing programming terminology.

4.3.2 Narrative, HUD and System Interface

To support the narrative-driven approach, the UI includes:

- **Dialogue System:** A dynamic text box with character portraits (e.g., Isyraq, The Corrupter) that displays story content. The system supports branching choices, allowing players to select responses that trigger different game events. Dialogue design is as shown in Figure 4.6.

Figure 4.6: Narrative Dialogue System displaying branching choices.

- **Quest UI:** A collapsible Heads-Up Display (HUD) that lists current objectives (e.g., "Meet Zafran," "Solve Loop Puzzle"). This ensures the player always has clear guidance on their learning goals. HUD design is as shown in Figure 4.7.

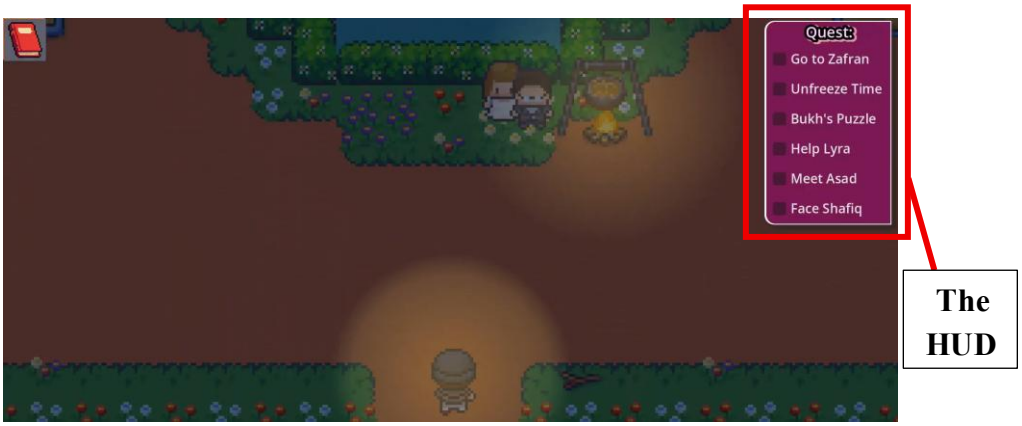


Figure 4.7: Collapsible Quest HUD tracking active learning objectives.

- **System Menus (Main, Pause, Credits):** To maintain an overall professional user experience (UX) flow, common game-state-management menus were implemented for the purpose of supporting that flow:
  - **Main Menu:** The Main Menu was created as the first entry point of the application and features a "Pixel Art" background of the world in which the game takes place to provide the immediate immersion of the player into the game's world. The main menu contains three of the most common navigation choices available for users; New Game, Load Game, and Options (which includes Volume Controls). UI design is as shown in Figure 4.8

Figure 4.8: The final version of Main Menu design for Code[X]: Nexus

- **Pause Menu:** Available through the ESC key, the Pause Menu will be using Godot's get_tree().paused = true function to freeze the entire game state at once. This means the student can pause the game at any time to break up their gaming experience without losing a crucial part of the narrative or timing out on puzzles. Menu design is as shown in Figure 4.9.

Figure 4.9: The final version of Pause Menu design for Code[X]: Nexus.

- **Credit Scene:** After completing the final segment of the World 4 Narrative Arc, the Credit Scene will scroll and display acknowledgement to the developer, all the assets utilized, and the tools used to create the software experience (Godot Engine).

Figure 4.10: The Credit Scene displaying acknowledgement after completing the game.

4.3.3 World and Puzzle Design

The world and puzzle design for this game were created with 2D pixel art graphics to give it a retro look and feel for the intended audience. the four worlds in the game have their own distinct color palettes and atmospheres to illustrate various parts of the computer system.

A. Environmental Theming

- **World 1:** Dark Woods (The Syntax Layer):

  Created using a dark, but greenish and brown forest style. A calm, non-threatening environment was designed so that new programmers can learn about basic movement and declare variables in this introductory environment. World design is as shown in Figure 4.13.

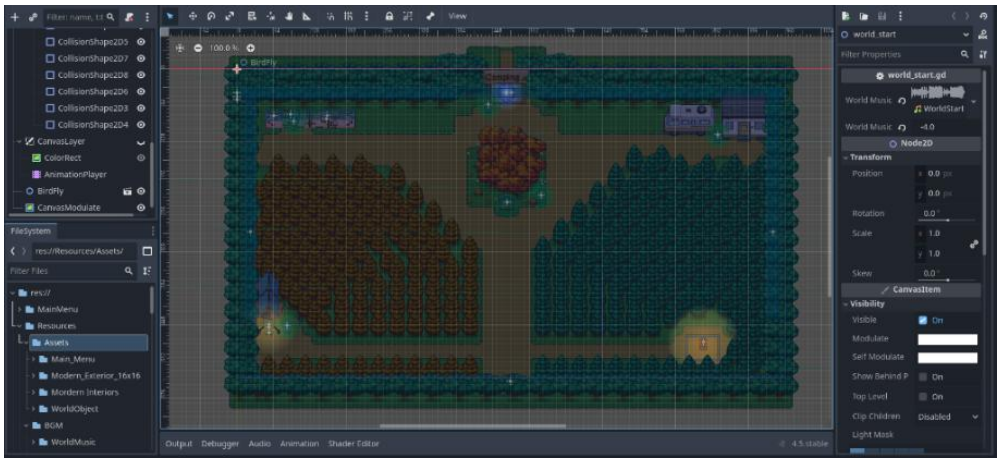


Figure 4.11: World 1 – Dark Woods Design.

- **World 2:** Zen Garden (The Logic Layer):

  An ordered and structured pathway and stone lanterns with bright lighting. The order is also reflective of the logic concepts (Loops and Conditional Statements) that are presented in this layer of the game, symbolizing clean and well-organized code. World design is as shown in Figure 4.12.

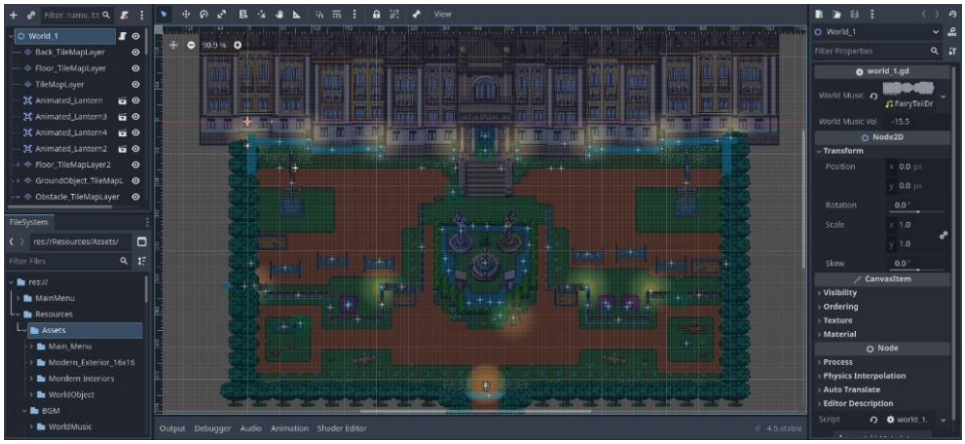


Figure 4.12: World 2 – Zen Garden Design.

- **World 3:** Zen Garden (The Function Layer):

  Dark, industrial with many machines and cores. This world represents the internal "function" of the program, and is visually represented by metal textures and a lower light level to indicate an increase in difficulty. World design is as shown in Figure 4.13.

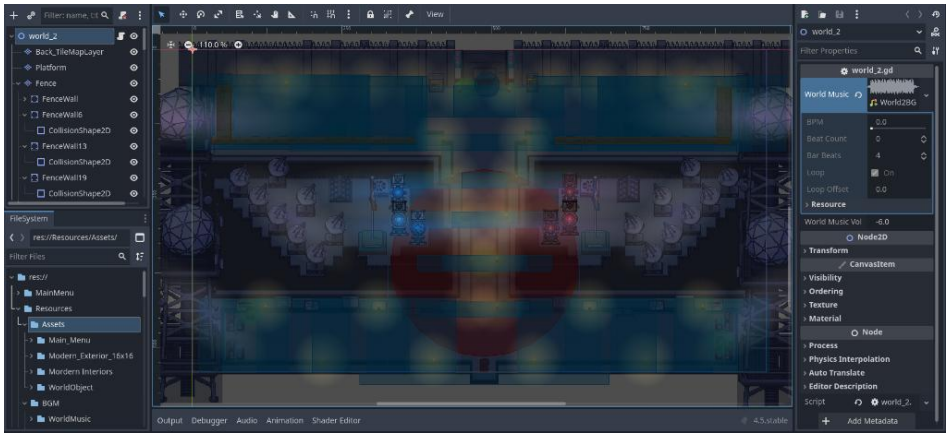


Figure 4.13: World 3 – Hollowed Core Design.

- **World 4:** Horizon's End (The Corrupted Layer):

  The most visually dramatic part of the game. The Red Tint created via CanvasModulate and the "Glitches" produced through custom screen shaders, represent the physical error of corrupted data and memory errors, which adds to the sense of urgency in the story. World design is as shown in Figure 4.14.

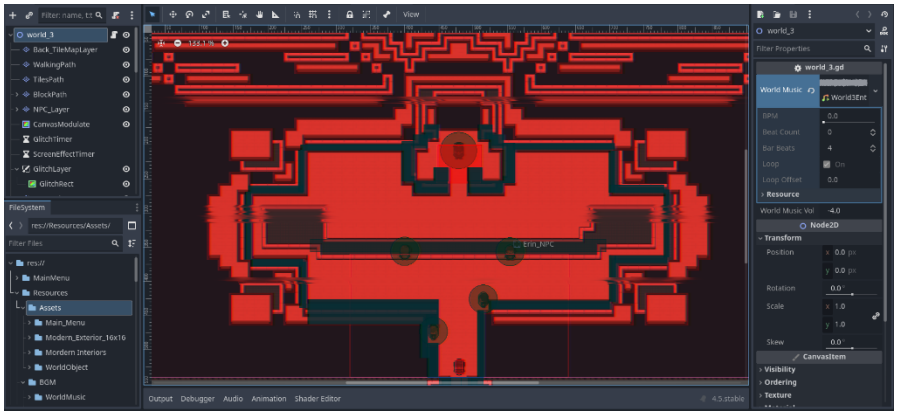


Figure 4.14: World 4 – Horizon's End Design.

B. Educational Puzzle Design Mechanics

To match student learning preferences and to eliminate as much cognitive overhead as possible due to learning how to program, the game uses three unique puzzle design strategies. These puzzle designs are incorporated directly into the story line so that coding does not feel like it is being tested but rather used as a gaming tool.

- **Interactive Dialogue Options (Concept Reinforcement):**

  Before players are asked to write code, they engage in Interactive Dialogue Options. This mechanic tests the player's conceptual understanding through conversation. Unlike standard RPG dialogues that only advance the plot, these interactions require the player to select the correct programming logic to proceed as shown in Figure 4.15.

  - **Mechanism:** The player will receive a problem posed by the NPC (e.g. "Stop this loop"), the player is presented with branching choices (e.g., "Use a break; statement" vs. "Delete the variable").
  - **Feedback:** If the player chooses incorrectly, they will be provided with a Corrective Dialogue from the NPC explaining the error, thus preventing them from continuing to develop on their incorrect understanding.

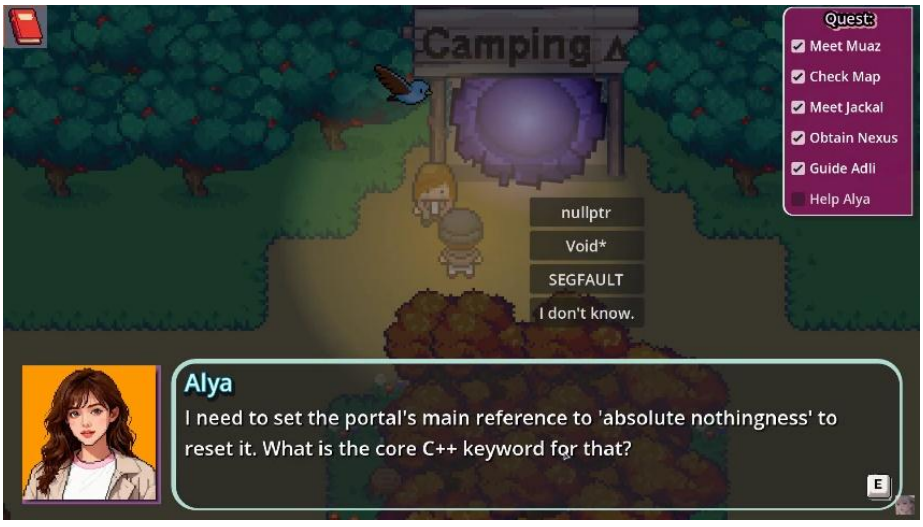


Figure 4.15: Interactive dialog option requiring the player to select the correct logic

- **Compiler Puzzle (Syntax Application)**

  The compiler puzzle is based on the core game loop which is a simulated IDE for the project's main objective; teach players C++ syntax as shown in Figure 4.16.

  - **Mechanism:** Players have to use "Broken Objects" (locked doors, broken robots) to open the Compiler UI. Players will be shown the "Broken Code" (red) and have to make the necessary edits in each line to resolve the syntax errors (for example, add a missing semicolon ";", change an integer to int).
  - **Validation:** The System validates player input by using a string parser (in CompilerUI.gd) to see if the syntax is correct. When the syntax is correct, the code changes from red to green, and the Broken Object is fixed.

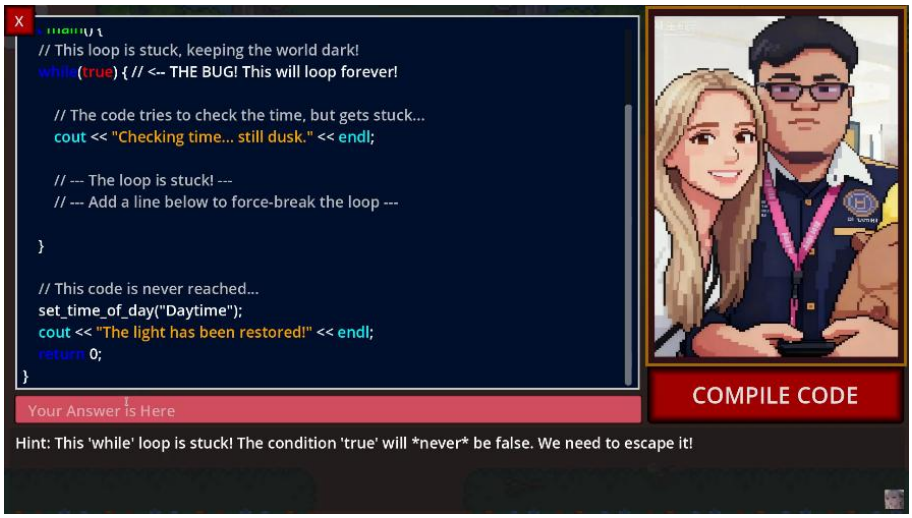


Figure 4.16: Interactive dialog option requiring the player to select the correct logic

- **Word Matching Puzzle (Terminology Association):**

  To teach terminology and data structures without the pressure of typing, the Word Matching Puzzle was implemented. This visual puzzle style is used primarily in World 2 (Zen Garden) to teach Data Types as shown in Figure 4.17.

  - Mechanism: Players are presented with a set of "Keys" (Values) and "Locks" (Definitions). For example, the player must match the value 10 to the int slot, or "Hello" to the string slot.
  - Pedagogy: This drag-and-select style mechanic helps build mental associations between data types and their literal values before the player is required to type them manually in the Compiler.

Figure 4.17: Word Matching Puzzle associating Data Types with corresponding values.

4.4 Character Design and Asset Implementation

A large cast of Non-Player Characters (NPCs) were created to further reinforce the narrative-based gameplay structure. The character artwork utilized a 2D Pixel Art style, which provided consistent visuals throughout the game to enhance its "retro-adventure" style based on the initial design specifications. There are over twenty different NPCs within the game; these can be grouped into two categories of their role in the narrative and their location within the game world. Each character serves not only as a part of the story but also as a "Learning Agent," providing players with hints, explaining programming syntax and assisting the player to advance through the curriculum.

4.4.1 NPC Categorization and Visual Style

The characters are distinctively designed to reflect the atmosphere of their respective worlds:

- **World 1 – Dark Woods:** A casual attired look to represent the everyday people of the game's beginning area, located in the Dark Woods world. Characters within this category include: Muaz, Adli, Alya, Jackal, and Maya. They provide introductory assistance to the player, enabling them to become familiarized with the game's fundamental movement and interaction mechanics. NPCs as shown in Figure 4.18.

Figure 4.18: Non-Playable Characters in Dark Woods

- **World 2 – Zen Garden:** Designed with a more formal or mystical aesthetic to represent knowledge keepers. NPCs like Zafran and Lyra are responsible for introducing logic structures (Loops and Conditionals), then NPC like Bukhari, Asad and Shafiq are responsible in testing the player's knowledge. NPCs as shown in Figure 4.19.

Figure 4.19: Non-Playable Characters in Zen Garden

- **World 3 – Hollowed Core:** Characters in this region, such as Aliff, Niko, Adriana, Zul and Aqil feature a new topic which is functions. This place represents the internal machinery of the system and world core, guiding the player through function-based puzzles and memory management tasks, then meeting upon The Corrupter, the one who caused it all. Making player reached out to defeat him. NPCs as shown in Figure 4.20.

Figure 4.20: Non-Playable Characters in Hollowed Core

- **World 4 – Horizon's End:** Due to the fact that the narrative culminates at Horizon's End, characters such as The Corrupter, Rapxstor, Isyraq, Akari, and Erin have all been designed with a distorted, "glitching" sprite effect. The visual representation of the critical system failure the player is required to fix, creates an elevated sense of urgency and raises the stakes for the final confrontation in the narrative. NPCs are in Figure 4.21.

Figure 4.21: Non-Playable Characters in Horizon's End

4.5 Build and Optimization Phase

After completing the Development Logic, the project progressed to the Build Phase. The Build phase included building the Godot Engine to generate a standalone executable format from the project to allow it to run as intended on each of the target platforms.

4.5.1 Export Configuration

The project was exported using Godot's standard export templates. Two specific build configurations were created:

- **Windows Desktop Build (.exe):**
  Compiled with 64-bit architecture for optimal performance and stability on all modern lab computer configurations. All debug symbols were removed to minimize file size and to make reverse engineering the project difficult as shown in Figure 4.22.

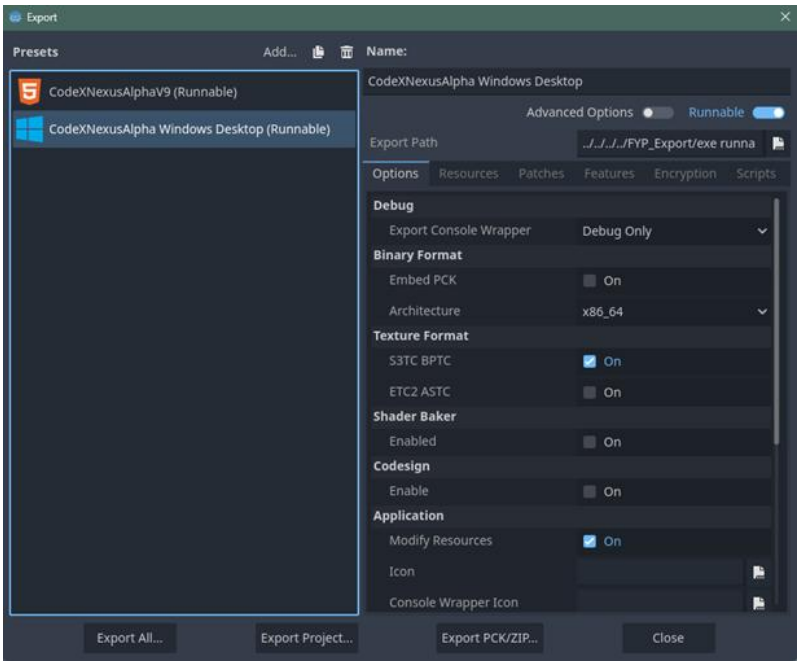


Figure 4.22: Windows (.exe) Godot Project Export Setting

- **HTML5 Web Build (WebGL):**

  Configured for browser-based play. Texture compression was set to VRAM Compressed to reduce load times over the internet as shown in Figure 4.23.

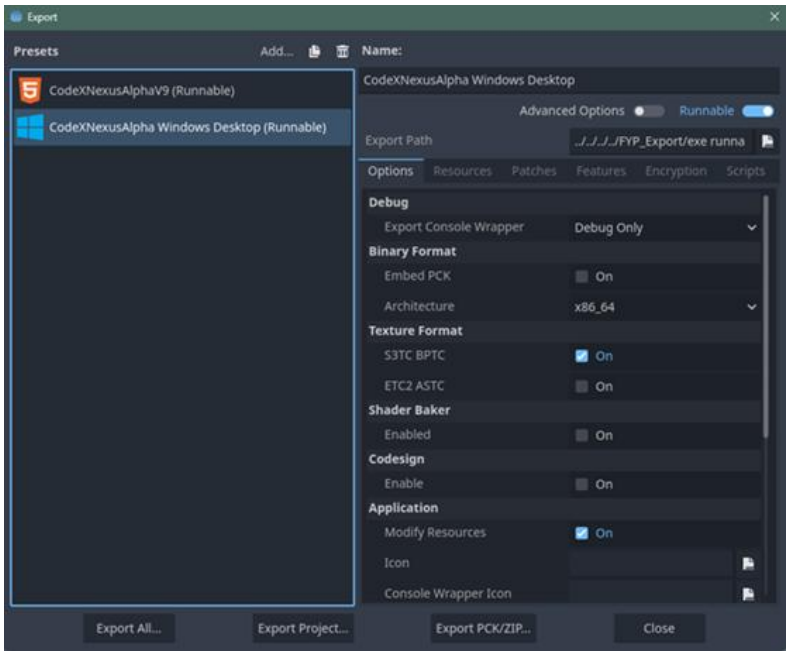


Figure 4.23: HTML5 Godot Project Export Setting

4.5.2 Asset Optimization

To adhere to the low-hardware requirements outlined in Chapter 3, asset optimization was performed prior to the final build:

- **Audio:**

  Background music files were converted from .wav to .ogg (Vorbis), reducing the total audio footprint by approximately 60% without perceptible quality loss as shown below.

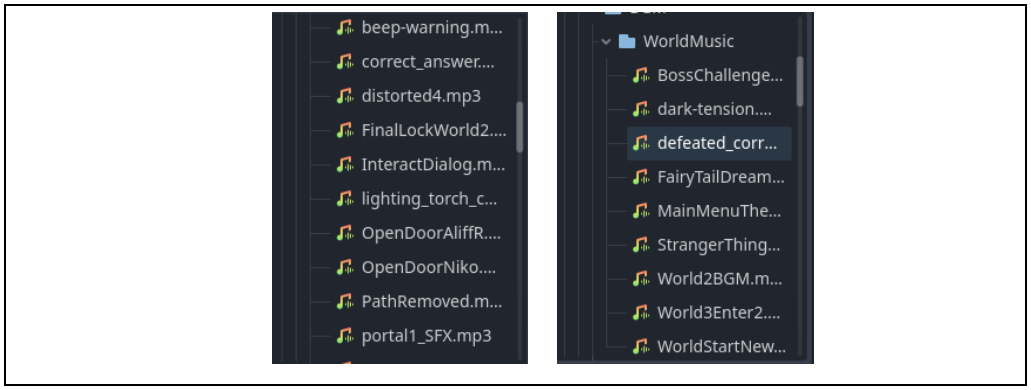


Figure 4.24: List of audios that were optimized

- **Scripts:**

  All GDScript files were compiled into binary .pck (Pack) files, ensuring faster execution speeds compared to raw text interpretation as shown in Figure 4.25.

index.apple-touch-icon.png	11,929	11,929	IrfanView PNG File
index.audio.position.worklet.js	2,973	1,141	JavaScript Sourc...
index.audio.worklet.js	7,298	2,179	JavaScript Sourc...
index.html	5,446	2,181	Brave HTML Doc...
index.icon.png	5,769	5,769	IrfanView PNG File
index.js	272,262	67,001	JavaScript Sourc...
index.pck	110,771,168	99,013,108	PCK File
index.png	21,443	19,266	IrfanView PNG File
index.wasm	36,160,334	9,790,044	WASM File

Figure 4.25: Godot game project exported into binary .pck files

4.6 Launch and Deployment Phase

Deployment of Code[X]: Nexus into a public hosting environment was the final stage of development of the game. For the purpose of this study, it was decided that itch.io would be the primary host for the game as it is accessible to independent developers and supports in-browser (HTML5) gameplay.

4.6.1 Platform Setup

A dedicated product page was created to host the game files. The page setup included:

- **Metadata:** Essential game information, including genre (Educational/Adventure) and input methods (Keyboard/Mouse), was configured to ensure proper indexing as shown in Figure 4.26.

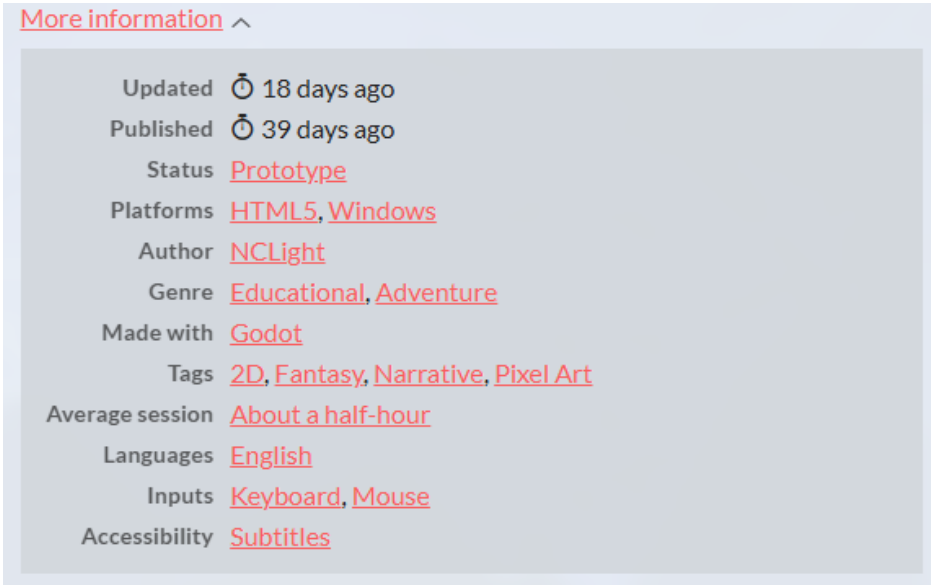


Figure 4.26: Metadata of the published game on Itch.io

- **Visual Assets:** Promotional screenshots and a gameplay trailer were uploaded to demonstrate the game's features to potential users as shown in Figure 4.27.

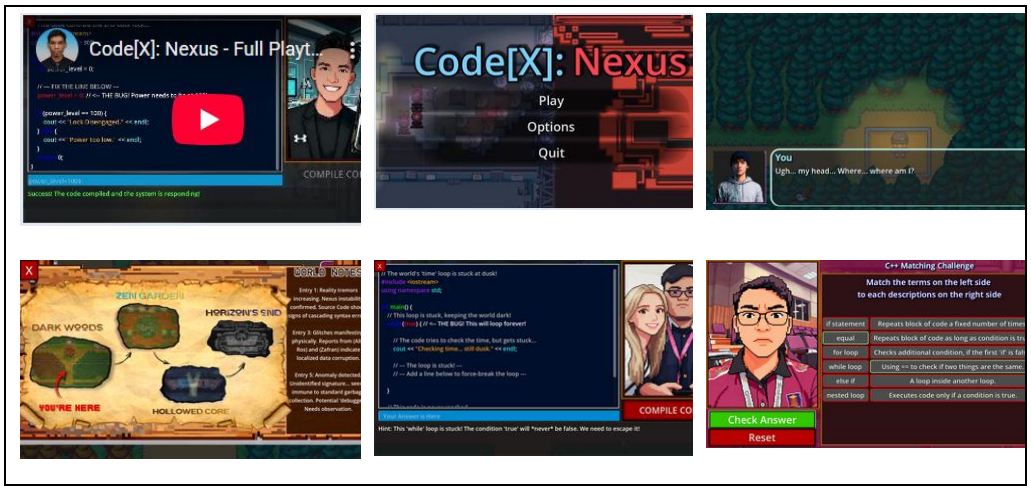


Figure 4.27 Gameplay and promotional screenshot presented in the Itch.io page

- **Licensing:** The project was released under a proprietary license ("No License / All Rights Reserved") to protect the intellectual property of the source code and assets, as shown in Figure 4.28.

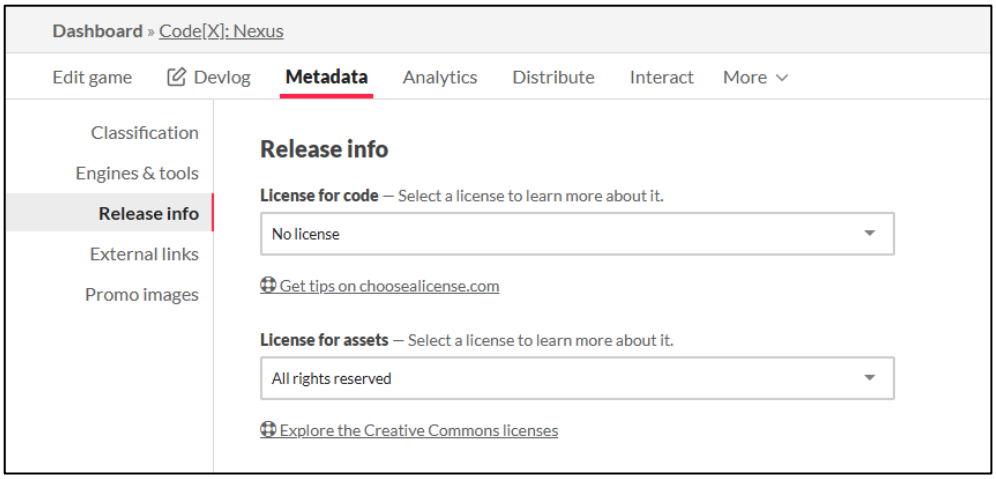


Figure 4.28: Licensing for code and assets setup in Itch.io page

4.6.2 Version Control and Patching

Upon initial launch, the game was labeled as version 1.0. The Itch.io "Butler" was used to keep track of builds so that patches could be made easily. This function helped a lot with testing by enabling developers to upload new builds quickly for fixes (i.e. physics collision fix; see section 4.5), rather than having all players have to download the whole game again. The process can be seen at Figure 4.29 and Figure 4.30.

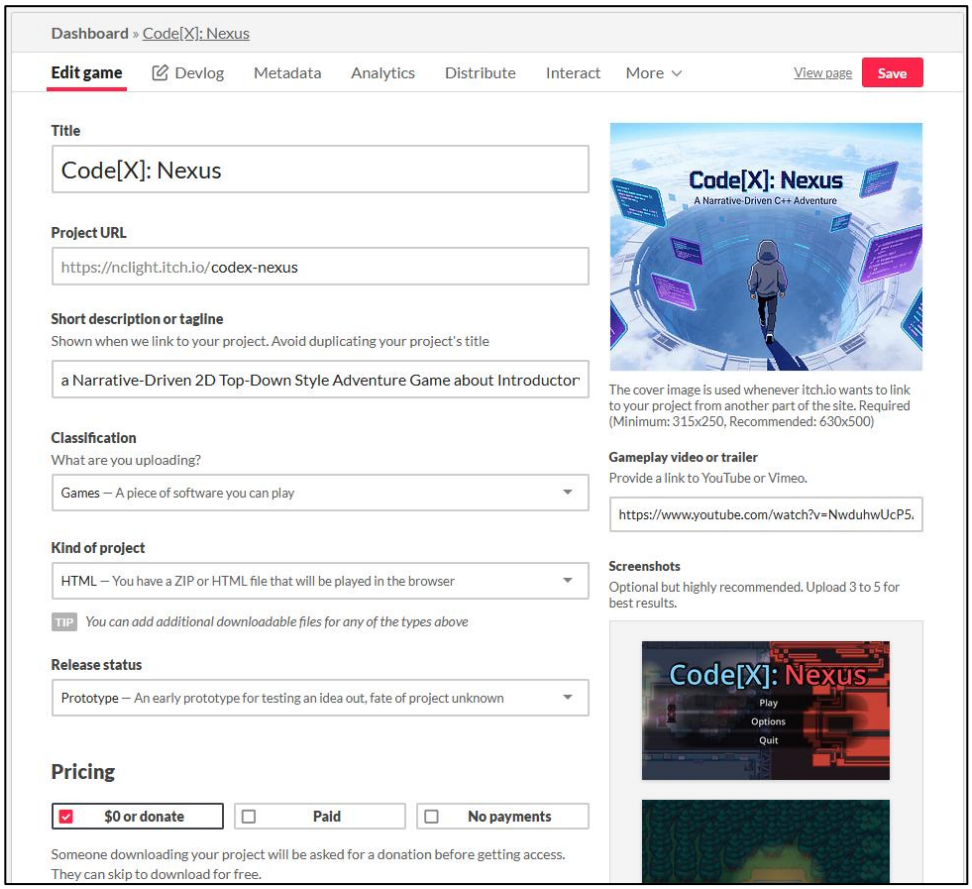


Figure 4.29: Itch.io main game page setup (part 1)

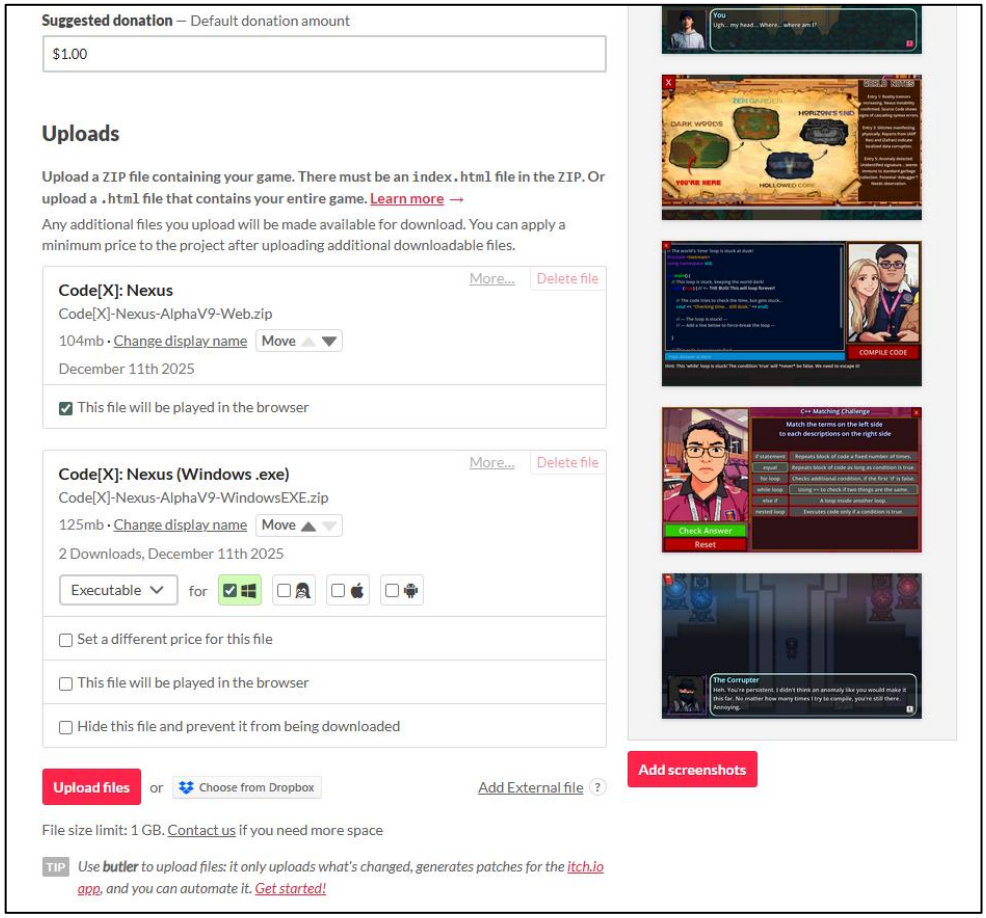


Figure 4.30: Itch.io main game page setup (part 2)

4.7 Testing and Evaluation

A combination of functional testing and preliminary user evaluation helped to ensure that the system met its intended technical and educational goals.

4.7.1 Functional Testing Results

Alpha testing was conducted to verify the stability of the game mechanics. Key findings included below with the information in the Table 4.1:

- **Puzzle Mechanics:** The input normalization algorithm (used for puzzle entry) worked well with a variety of valid code formats, thus eliminating the source of much player frustration generated by slight variations in spacing when entering the same code.
- **Progression Blocks:** There were two major bugs at the beginning of testing; one bug allowed players to continue moving while the game displayed cut scenes, and another bug allowed players to bypass the required prerequisites before completing each quest. Both bugs were fixed via the use of the state machine logic in the Quest Manager.
- **Platform Compatibility:** The game successfully built and ran on both Windows (in .exe format), and in HTML5 formats. The WebGL build needed optimizations specifically for physics triggers which was solved by manually polling for collisions.

Table 4.1: Summary of Identified Bugs and Resolutions during Functional Testing

Bug ID	Issue Description	Severity	Resolution Status
B – 01	Player stuck in wall collision in World 2	High	Fixed (Adjusted CollisionShape2D)
B – 02	Compiler accepts empty input as correct	Medium	Fixed (Added null check validation)
B – 03	Player input remained active during dialogue sequences	High	Fixed (Implemented State Machine to freeze player)
B – 04	Z-Index rendering issues (Player appearing behind ground objects)	Low	Fixed (Enabled Y-Sort on all World Scene nodes)
B – 05	Quest progress resetting after scene transition	Critical	Fixed (Implemented Persistent Singleton 'QuestManager')

4.7.2 User Experience (UX) and Learning Evaluation

A quantitative survey was conducted to evaluate the usability, engagement, and educational impact of Code[X]: Nexus. A total of 31 respondents participated in the study. The feedback was collected via Google Forms after the participants had completed at least the first chapter of the game.

A. Demographic Analysis

The respondents consist mainly of students who fit the targeted age range for a first-year diploma program. It can be seen from Figure 4.31 that 48.4 % of all participants were 19-20 years old; 25.8 % were 21-22 years old. Therefore, there is no doubt that the test subjects reflect the targeted audience of the project.

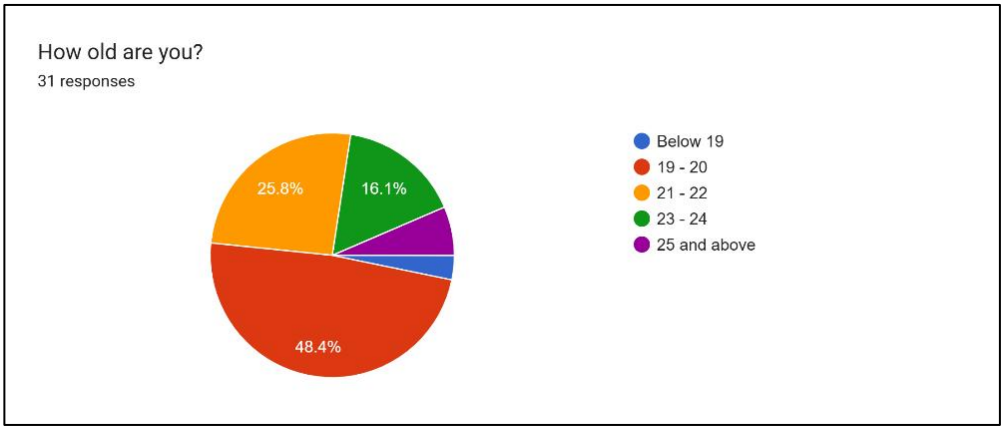


Figure 4.31: Age distribution of the 31 survey respondents.

Regarding gaming habits, the participants showed a diverse range of experience. Figure 4.32 indicates that 35.5% play video games "Once in a while," while 32.3% play "A few times a week." Only a small fraction indicated they "Almost never" play games. This suggests that the majority of the test group had some level of digital literacy regarding game mechanics.

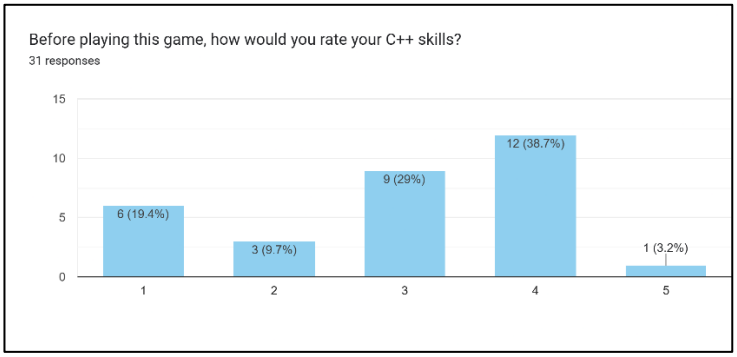


Figure 4.32: Frequency of video game usage among respondents.

To assess the effectiveness of the learning material, participants were asked to rate their C++ skills prior to playing. As illustrated in Figure 4.33, the results were mixed, with 38.7% rating their skills as a '4' (Good), while 19.4% rated themselves as '1' (Beginner). This variety provided valuable insight into how the game caters to both complete novices and those with some prior knowledge.

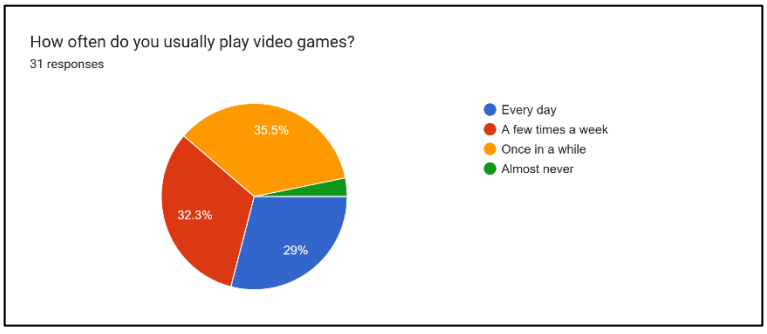


Figure 4.33: Self-reported C++ skill level before playing Code[X]: Nexus.

B. Usability and Learning Outcomes

After analyzing the demographics of the respondents, the survey tested the respondents' immediate reactions to the gameplay mechanics. The reaction to the control system is documented in Figure 4.34. In total, 74.2 % of respondents rated the control system as a 4 (easy to operate); 16.1 % rated the control system as a 5 (perfectly easy to operate). It may therefore be assumed that the majority of the targeted audience found the WSAD movement and interaction systems intuitive.

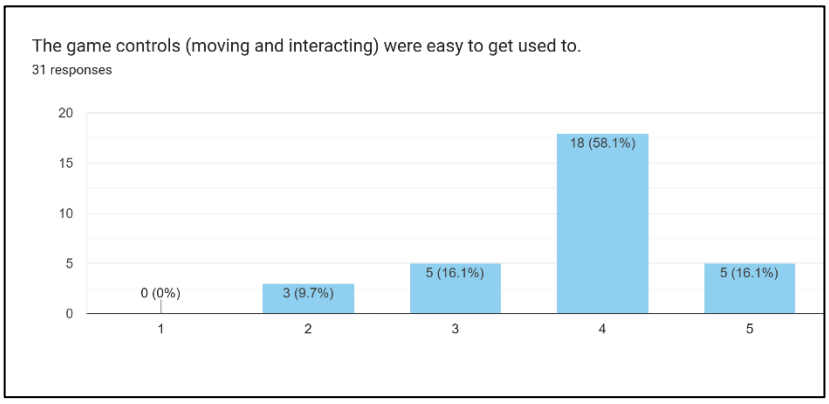


Figure 4.34: User rating for game controls intuition (Movement and Interaction).

Similarly, the User Interface (UI) was highly rated by respondents. Figure 4.35 documents that 41.9 % of users rated the UI as easy to understand (rating 4); 32.3 % rated it as best possible. It therefore appears that the design decision to keep the HUD as minimalistic as possible and to make the compiler window look similar to a standard text editor was justified.

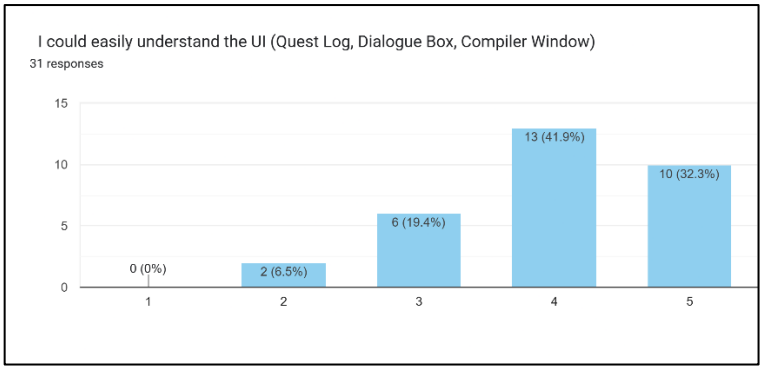


Figure 4.35: User rating for UI clarity and navigation.

For educational software, technical stability is just as important as usability. The question of whether the game ran smoothly without major bugs elicited a very positive response. As shown in Figure 4.36, 48.4 % of respondents completely agreed (rating 5) that the game ran smoothly; 38.7 % agreed (rating 4). It can therefore be concluded that the optimizations made during the build phase (Section 4.5) were successful.

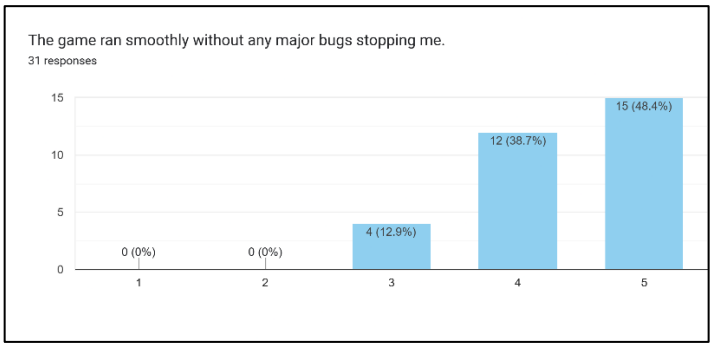


Figure 4.36: User feedback on technical stability and bug frequency.

Finally, the respondents were asked to evaluate the visual aspect of the game so that the "retro-pixel" style of the graphics would appeal to the target audience. Figure 4.37 demonstrates a clear preference for this art style. In fact, 90.4 % of respondents rated the pixel art direction as a 4 (good) or 5 (excellent). The average rating of 4.32 furthermore indicates that the pixel art direction was able to maintain the players' interest and thus contributed to the overall atmosphere of the game.

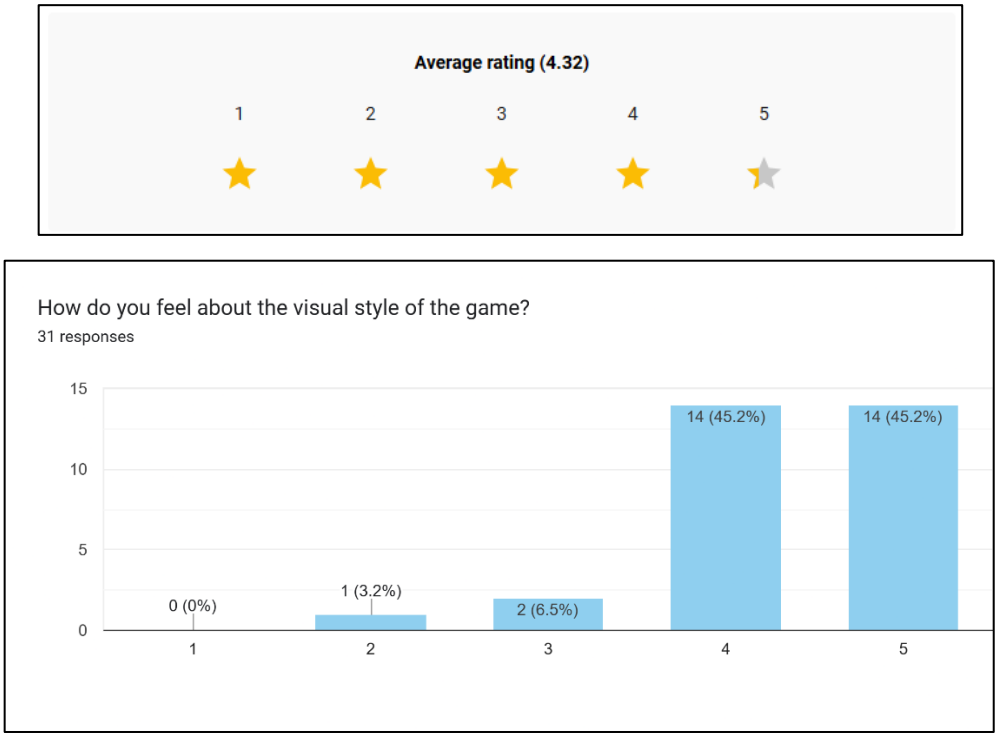


Figure 4.37: User satisfaction with the game's visual style.

C. Educational Effectiveness and Motivation

The core objective of Code[X]: Nexus was to validate the effectiveness of Game-Based Learning (GBL) in teaching C++. The survey results provide strong evidence supporting this objective.

Respondents were asked if the "Compiler Puzzles", which are an interactive part of the game, presented a better explanation of the concepts than reading textbooks. As demonstrated in Figure 4.38, the responses were very positive. In total, 90.3 % of the respondents indicated that they agreed (rating 4) or completely agreed (rating 5) that the game mechanisms enabled them to see syntax errors much better than static text.

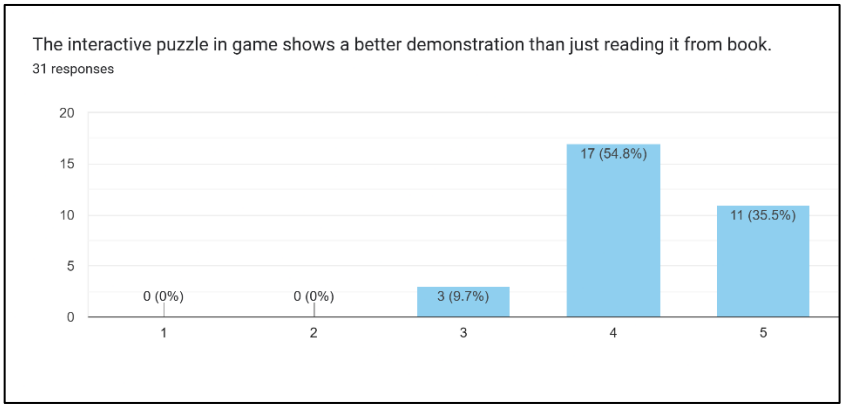


Figure 4.38: Participant agreement on the effectiveness of interactive puzzles vs. textbooks.

Furthermore, the narrative delivery of the educational content was also successful. Figure 4.39 demonstrates that 96.8 % of the players felt that the dialogues with the NPCs (Isyraq and Zul) explained the programming concepts clearly. This supports the design decision to implement "learning agents" in the story as a way to explain complex topics such as loops and functions.

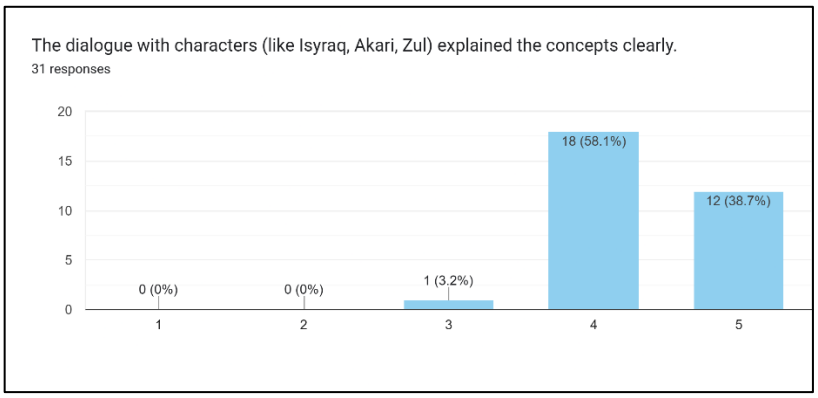


Figure 4.39: User rating on the clarity of NPC dialogue in explaining C++ concepts.

Regarding the outcome of the learning process, the project attempted to increase the self-confidence of the students regarding the basics of C++. When asked if they felt more confident in variables, loops, conditionals etc., after playing the game, 80.6 % of the respondents rated their increased confidence as 4 or 5 (see Figure 4.40). This indicates that the game was able to bridge the gap between theoretical knowledge and practical experience.

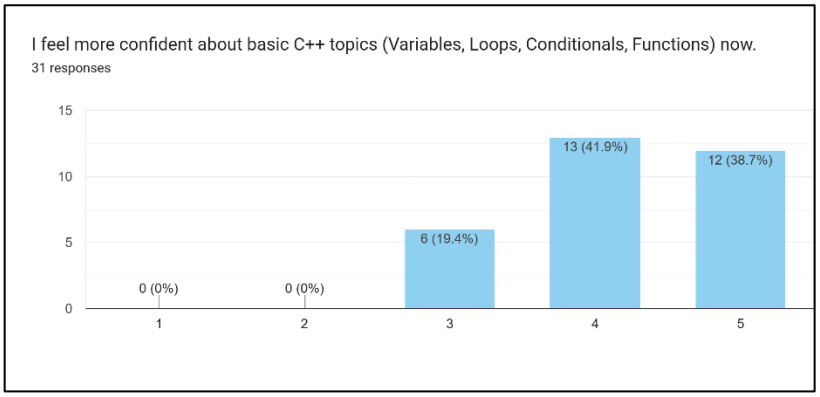


Figure 4.40: Self-reported increase in confidence regarding C++ topics after gameplay.

Finally, the survey measured student motivation, a critical factor in GBL. Figure 4.41 highlights the most significant finding: 67.7% of participants gave the highest possible rating (5) when asked if they would prefer playing a game like this over listening to a standard lecture. This data strongly supports the hypothesis that gamification can significantly enhance student engagement in technical subjects.

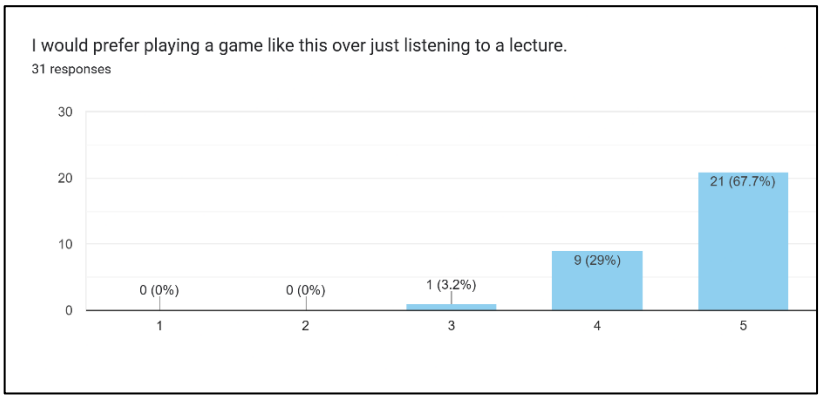


Figure 4.41: Student preference for Game-Based Learning compared to traditional lectures.

D. Narrative and Game Mechanics

To evaluate the impact of the "Narrative-Driven" approach (Objective 2), participants were asked if the story motivated them to complete the game. As seen in Figure 4.42, the results were positive, with 67.7% of respondents rating the narrative's motivational factor as high (Ratings 4 and 5). This indicates that the plot involving "The Corrupter" successfully provided the necessary context to keep students engaged with the learning material.

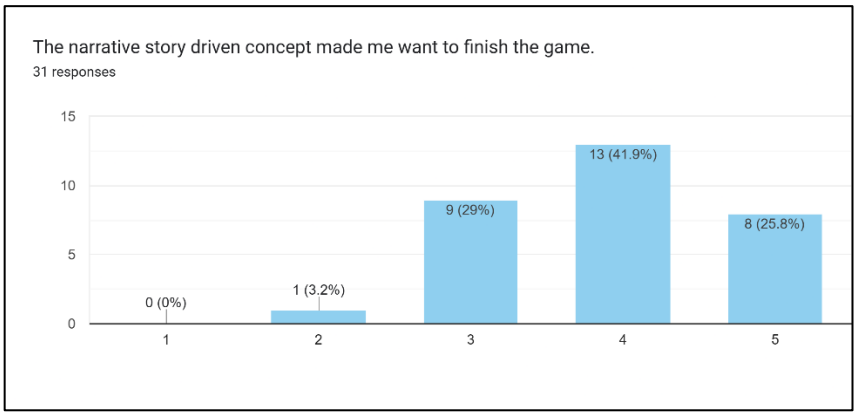


Figure 4.42: User rating on whether the narrative story motivated them to finish the game.

Atmospheric Design (Pixel Art, Lighting and Audio) received the most praise throughout the study. Figure 4.43 documents that all respondents rated the atmosphere as 4 or 5. Specifically, 54.8 % of respondents awarded a full 5. This complete approval of the aesthetic choices (e.g. the glitch effects in World 3) of the game confirms that the design decisions concerning the immersion of the learning environment were very successful.

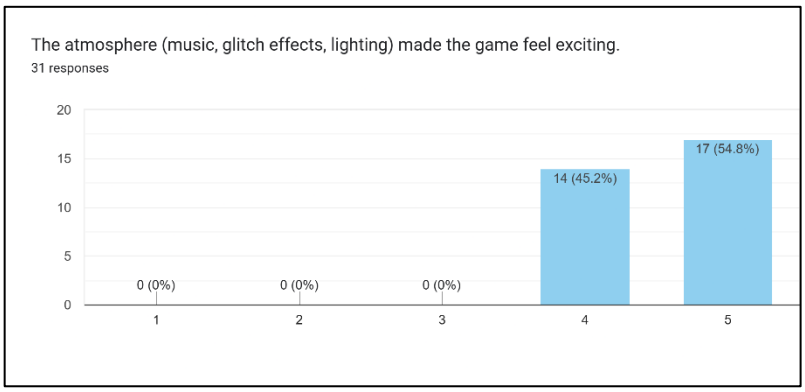


Figure 4.43: User rating on the game's atmosphere and excitement level.

Additionally, the respondents were asked how difficult the puzzles were to solve, in order to determine whether the puzzles were too frustrating for the students or too trivial. According to Figure 4.44, 83.8 % of the respondents felt that the puzzle difficulty was "Just Right" (ratings 4 and 5). Therefore, it seems that the progression of the game difficulty curve, starting with simple variable declarations and ending with complex loops, was tuned to the students' individual learning pace and, consequently, achieved the necessary "Flow" state for GBL.

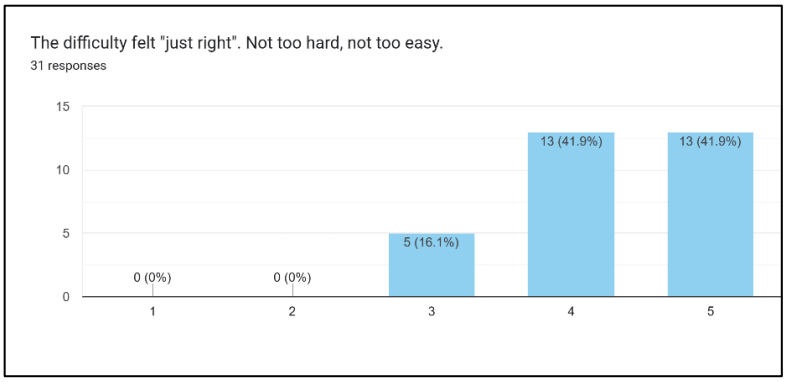


Figure 4.44: User perception of game difficulty balance.

E. User Preferences and Final Feedback

In the final section of the survey asked the respondents which environment of the game they liked best. According to Figure 4.45, there is a large preference for the "Horizon's End (World 4)" environment, with 61.3 %. This is due to the fact that World 4 is the climax of the story, where the "glitch" aesthetics and the narrative tension are at their peak. On the other hand, the "Dark Woods (World 1)" environment received the least amounts of preferences. This is likely because it is mostly used as a tutorial area and has simpler mechanics.

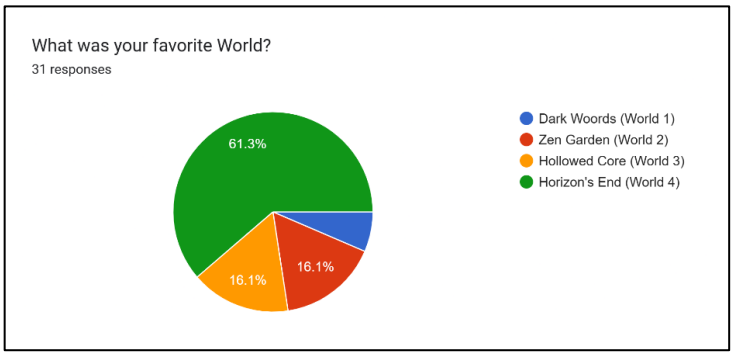


Figure 4.45: User preference for different game worlds.

Perhaps the most critical finding regarding the educational design comes from the feedback on puzzle clarity. When asked, "Was there any part of the coding puzzles that felt too confusing?", 100% of respondents answered "No" (Figure 4.46). This unanimous response is a significant achievement for an educational tool. It confirms that the scaffolding technique which gradually introducing concepts through dialogue before presenting the actual code, successfully mitigated the frustration often associated with learning C++.

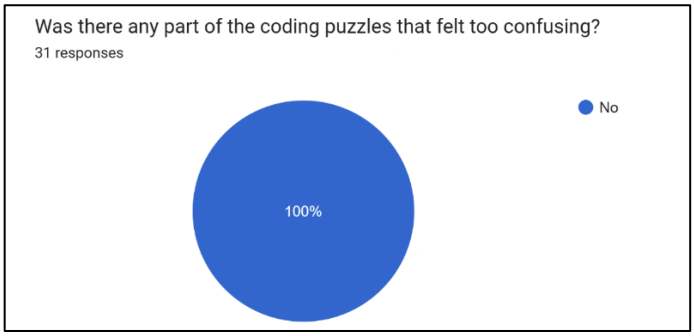


Figure 4.46: Percentage of users who encountered confusing puzzles.

4.8 Discussion

The development of Code[X]: Nexus demonstrates that abstract programming concepts can be effectively gamified. By mapping C++ topics to gameplay mechanics (e.g., using "Loops" to fix a recurring weather bug, or "Variables" to unlock a door), the game successfully creates a contextual learning environment.

The use of a narrative-driven structure proved essential. As noted in the literature review, storytelling increases emotional investment. In this project, the "Corrupter" served as an antagonist that personified "bugs" and "errors," giving the player a tangible reason to master debugging skills. This contrasts with traditional learning, where errors are simply obstacles without context.

One challenge encountered was balancing the difficulty of the puzzles with the flow of the game. Early testing showed that strict syntax requirements could halt progress. The implementation of the "Hint System" and "Code Normalization" was a critical adjustment to maintain the "Flow" state, preventing frustration while ensuring learning standards were met. This directly satisfies Objective 2, as evidenced by the functional modules presented in Section 4.2.

4.9 Summary

This chapter detailed the successful implementation of Code[X]: Nexus. The project delivered a fully functional 2D adventure game featuring a robust quest system, interactive C++ learning puzzles, and a cohesive narrative. The technical algorithms for state management and event handling ensured a stable user experience, while the UI design supported clear educational feedback. Testing results indicate that the game is a viable tool for supplementary Computer Science education, effectively bridging the gap between theoretical study and interactive practice.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Introduction

This project goal is to address the challenges of low student engagement and difficulty in grasping abstract C++ programming concepts among the first-year Computer Science university students. By developing Code[X]: Nexus, a narrative-driven 2D adventure game, it aimed to provide an alternative learning tool that integrates Game-Based Learning (GBL) principles with the UiTM Diploma Computer Science curriculum. This chapter summarizes the project's achievements in relation to its initial objectives in Chapter 1, highlighting the limitations encountered during development and proposed suitable recommendations for future enhancements of the project.

5.2 Conclusion

The development and evaluation of Code[X]: Nexus have successfully met the three main objectives outlined at the beginning of this study:

Objective 1: To identify an effective Game-Based Learning (GBL) method in programming education in enhancing student motivation.

As part of the literature review and the development of this project it was found that using a combination of Scaffolding Puzzles and Narrative Visualization would be very effective at teaching C++. The research further confirmed what has been previously noted, that when abstract concepts such as loops and variables are presented in isolation to students, they tend to struggle with them. However, by recontextualizing those abstract concepts into game-based mechanics (i.e., how a "loop" can be used to repair a malfunctioning weather system), an effective GBL method was created that could reduce the students' cognitive load while increasing their level of engagement.

Objective 2: To develop a 2D top-down style learning adventure game, in learning programming concept that includes GLB elements using Godot Engine v4.4.

Objective 2 was completed in its entirety as the result of the creation of Code[X]: Nexus. The final application is a complete, working, feature-rich game that can be deployed onto both Windows and Web Platforms. Some of the key technical accomplishments were the creation of a robust Quest System and State Manager that will allow the tracking of a student's progression across four different game worlds, a custom Compiler UI that will parse and validate real C++ syntax and simulate a simple IDE environment, and the use of "retro" pixel-art aesthetic and atmospheric audio to create an immersive learning experience all created within the Godot 4 Ecosystem.

Objective 3: To assess the usability of the developed GLB game towards the targeted students.

User Acceptance Testing (UAT) with 31 participants confirmed the game's educational value. The results indicated that 90.3% of students found the interactive puzzles more effective than traditional textbooks and 100% reported zero confusion with the puzzle logic. Furthermore, the strong preference for the narrative climax in World 4 demonstrated that storytelling is a powerful driver for student motivation, validating the hypothesis that narrative-driven GBL significantly improves engagement.

5.3 Limitations of the Project

While the project was very successful, there are also several limitations we found while developing the project and testing it:

1. **Scope of Curriculum:** Curriculum Range: The game currently only has an introduction section for the following C++ basics; Variables, Loops, Conditionals, and Basic Functions. Time span given to develop the game are not enough to cover advanced C++ topics such as Pointers, Arrays and Object-Oriented Programming (OOP).
2. **Platform Optimization:** Optimization for Platforms: Although the Windows version is stable, the Web version (HTML5) experienced some minor issues with the physics, which can be caused by different browsers. Therefore, some optimizations had to be performed in order to limit the number of physics-based puzzles possible in the web build. These optimizations could have reduced the complexity of these types of puzzles in the browser build.
3. **Assessment Depth:** Currently, the assessment system will provide players with immediate feedback (Correct or Incorrect), but does not include a detailed analytical dashboard to help the instructor. Instructors currently do not have the ability to see the number of attempts a student took to solve a puzzle, in addition to other information, to assist in using the system as a formal assessment tool.

4. **Content Volume:** The length of the gameplay experience is roughly 30-45 minutes. Although this length is suitable as a supplemental tool, it is not nearly long enough to replace all of the lab exercises of an entire semester.

5.4 Recommendations for Future Work

To maximize the full capabilities of Code[X]: Nexus as an all-encompassing learning tool, the following recommendations have been identified to support future development:

1. **Expansion of Curriculum:** Future versions of Code[X]: Nexus will need to include "Act II" which is intended to cover intermediate topics (such as Arrays, Pointers and Structs). Additional game elements such as an "Inventory System" representing Arrays could be added to help students visualize these concepts.
2. **Instructor Dashboard:** A back-end database utilizing either Firebase or SQL can be used to capture student performance metrics. This feature would enable instructors to log into the system and review how their entire class is performing; also, this would enable them to identify those students that are having difficulty with specific concepts.
3. **Mobile Porting:** To deploy Code[X]: Nexus for mobile devices (Android and iOS), it would be necessary to optimize the controls for touch screen usage. This would provide increased access to the tool for students to learn programming and practice coding on tablets and smartphones outside of computer lab settings.
4. **Multiplayer Leaderboard Features:** By adding a global leaderboard for "Optimization Challenges", such as solving a code puzzle at the fastest time possible, would add a competitive aspect and by doing so, provide additional motivation for students to participate via social gamification.

5.5 Summary

In conclusion, Code[X]: Nexus stands as a proof-of-concept that abstract programming concepts can be effectively taught through narrative-driven gameplay. By bridging the gap between theoretical computer science and interactive entertainment, this project provides a viable, engaging and educational alternative to traditional teaching methods. The positive reception from the target demographic suggests that tools like Code[X]: Nexus have a significant role to play in the future of technical education. The results of this research show that utilizing puzzle-based lessons in curriculum with compelling storyline has proven to be effective at lowering learning anxiety and increasing confidence in students. These results along with the positive response to the target demographic indicate that there will likely be a continued growth of educational tools such as Code[X]: Nexus to help create a more engaging experience in the classroom for the future of technical education.

References

Al-Maroof, I. A. J., Al-Emran, M., Al-Shadafan, S., & Al-Ahbabi, S. (2024). A review of game-based learning in computer science education. International Journal of Information Management Data Insights, 4(1), 100220.

Alves, J. R. M., & Letouze, P. (2018). The curriculum integration of a course of "Introduction to Programming Logic" with a serious game – Colobot. International Journal of Social Science and Humanity, 8(11), 275–280. https://doi.org/10.18178/IJSSH.2018.V9.974

Arnedo, J., & García Solórzano, D. (2023). Coding is fun: Engaging adult online learners using programming games. In A. García Holgado & F. J. García Peñalvo (Eds.), Proceedings TEEM 2022 (pp. 606–614). Springer. http://doi.org/10.1007/978-981-99-0942-1_63

Borrás-Gené, O., Hijón-Neira, R., Paredes-Barragán, P., & Serrano-Luján, L. (2024). A hybrid escape room to foster motivation and programming education for pre-service teachers. International Journal of Game-Based Learning, 14(1), 1–17.(https://doi.org/10.4018/IJGBL.343525)

Boyle, E. A., Hainey, T., Connolly, T. M., Gray, G., Earp, J., Ott, M., ... & Pereira, J. (2016). An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games. Computers & Education, 94, 178–192. https://doi.org/10.1016/j.compedu.2015.11.003

Chan, H. L., Lin, H. C., & Lee, T. H. (2023). Editorial: New educational technology and its impact on student-centered learning. Frontiers in Psychology, 14, 1189423.

Cheong, C., Flippou, F., & France, D. (2020). Game-based learning in computing education: A mapping review. ACM Computing Surveys, 53(4), 1–37. https://doi.org/10.4018/IJGBL.2020010101

Choi, I. C., Choi, W. C., & Chang, C. I. (2024, November). Exploring the impact of CodeCombat Python programming curriculum on student motivation at primary school. *In 2024 8th International Conference on Education and E-Learning (ICEEL)* (pp. 138-143). ACM. https://doi.org/10.1145/3719487.3719521

Choi, W. C., & Choi, I. C. (2024). Investigating the effect of the serious game CodeCombat on cognitive load in Python programming education. In Proceedings of the 2024 IEEE World Engineering Education Conference (EDUNINE) (pp. 1–10). IEEE. https://doi.org/10.1109/EDUNINE60625.2024.10500551

Di Nardo, V., Fino, R., Fiore, M., Mignogna, G., Mongiello, M., & Simeone, G. (2024). Usage of gamification techniques in software engineering education and training: A systematic review. Computers, 13(8), 196. https://doi.org/10.3390/computers13080196

de Oliveira, P. F. F., de Oliveira, A. L. M., & de Souza, W. S. L. (2022). Gamification in programming education: A systematic mapping study. Journal of the Brazilian Computer Society, 28(1), 1–28.

Dobroskok, I., Tan, A., Bekirogullari, Z., Kurucay, M., & Ivanova, T. (2022). Game development software tools in higher educational institutions: Experience of Ukraine, Turkey and Bulgaria. ResearchGate. https://www.researchgate.net/publication/365574044

Giannakoulas, V., & Xinogalos, S. (2024). When Narratives Meet Algorithms: How Story-Based Educational Games Transform Learning into a Computational Adventure. GESS Education.

Heithausen, C. (2020, February 15). A look at "Human Resource Machine" according to Papert's ideas. Cologne Game Lab, TH Köln – University of Applied Sciences. https://www.researchgate.net/publication/361112910

Herlambang, A. D., Ramadana, M. R., Wijoyo, S. H., & Phadung, M. (2024). Students' cognitive load on computer programming instructional process using example-problem-based learning and problem-based learning instructional model at

vocational high school. Elinvo (Electronics, Informatics, and Vocational Education), 9(2), 309–320. https://doi.org/10.21831/elinvo.v9i2.57882

Holly, S., Tan, K. L., & Lim, W. Y. (2024). Gamification in programming education: A systematic review. IEEE Transactions on Education, 67(2), 123–135. https://doi.org/10.48550/arXiv.2406.03055

Ishaq, K., Alvi, A., ul Haq, M. I., Rosdi, F., Choudhry, A. N., Anjum, A., & Khan, F. A. (2024). Level up your coding: A systematic review of personalized, cognitive, and gamified learning in programming education. PeerJ Computer Science, 10, e2310. https://doi.org/10.7717/peerj-cs.2310

Iudean, B. (2024). Storytelling as a pedagogical tool in computer science education: A case study on software systems verification and validation. In Proceedings of the 16th International Conference on Computer Supported Education (CSEDU 2024) (Vol. 2, pp. 557–564). SCITEPRESS. https://doi.org/10.5220/0012687300003693

Kroustalli, C., & Xinogalos, S. (2021). Studying the effects of teaching programming to lower secondary school students with a serious game: A case study with Python and CodeCombat. Education and Information Technologies, 26(5), 6069–6095. https://doi.org/10.1007/s10639-021-10596-y

Kalogiannakis, K., Papadakis, S., & Zourmpakis, A. I. (2021). Gamification in science education. A systematic review of the literature. Education Sciences, 11(1), 22. https://doi.org/10.3390/educsci11010022

Kunovic, I., Filipovic, I., & Sovic Krzic, A. (2024). Teaching robotics concepts with Besiege: A game-based approach for developing design and problem-solving skills. In 2024 47th MIPRO ICT and Electronics Convention (MIPRO) (pp. 1472-1477). IEEE. (https://doi.org/10.1109/MIPRO60963.2024.10569275)

Kucher, T. (2021). Storytelling and decision-making in educational games: Enhancing memory recall and engagement. Journal of Educational Technology Development and Exchange, 14(1), 45–60.

Lozano-Lozano, C., Cárdenas-Robledo, L. A., & Gonzalez-Ramirez, T. (2023). A serious game to learn the OSI model based on experiential learning. Computer Applications in Engineering Education, 31(4), 932-953.

Soleymani, A., De Laat, M., & Specht, M. (2025). Evaluating value creation, motivation, and personal experiences in a game-based professional learning network for computer science education. Electronic Journal of e-Learning, 23(3), 45-63. https://doi.org/10.34190/ejel.23.3.3757

Qian, M., & Clark, K. R. (2019). Game-based learning and 21st-century skills: A review of recent research. Educational Research Review, 27, 1–16. https://doi.org/10.1016/j.chb.2016.05.023

Tan, S. L., Neill, S., & D'Souza, D. (2021). Motivation and learning in gamified computer science courses: An exploratory study. Computers & Education Open, 2, 100050.

Tene, T., Vique López, D. F., Valverde Aguirre, P. E., Cabezas Oviedo, N. I., Vacacela Gomez, C., & Bellucci, S. (2025). A systematic review of serious games as tools for STEM education. Frontiers in Education, 10, 1432982. https://doi.org/10.3389/feduc.2025.1432982

Tian, R., Saito, D., Washizaki, H., Fukazawa, Y., Kobayashi, H., & Tsuji, A. (2024). Enhancing programming education through game-based learning: Design and implementation of a Puyo Puyo–inspired teaching tool. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024) (Vol. 2, pp. 1838–1839). ACM. https://doi.org/10.1145/3626253.3635477

van Gaalen, A. E. J., Brouwer, J., Schönrock-Adema, J., Bouwkamp-Timmer, T., Jaarsma, A. D. C., & Georgiadis, J. R. (2021). Gamification of health professions education: A systematic review. Advances in Health Sciences Education, 26, 683-711. https://doi.org/10.1007/s10459-020-10000-3

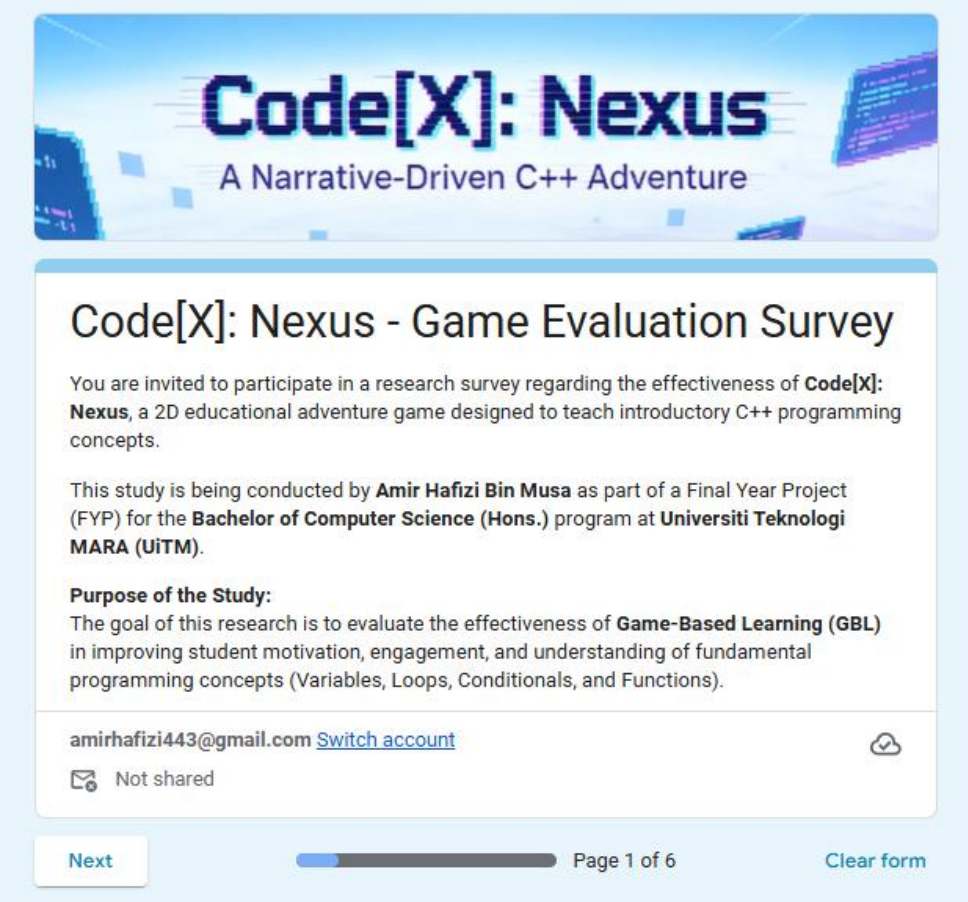
Zainuddin, Z., Chu, S. K. W., & Shujahat, M. (2020). Mastery learning in gamified classrooms: A systematic literature review. *Education and Information Technologies*, *25*(5), 4525–4549.

APPENDIX

APPENDIX A: TIMELINE FOR THE PROPOSAL PROJECT

Phase	Activity	Start Date	End Date
FYP 1: Planning & Analysis	Topic Proposal & Selection	2025-03-01	2025-04-01
	Literature Review	2025-04-01	2025-05-15
	Methodology Determination	2025-05-15	2025-06-15
	Requirement Gathering	2025-06-15	2025-07-15
	Ethics Approval (FERC/BERC)	2025-07-01	2025-07-18
	Submission of Report (Milestone)	2025-07-01	2025-07-10
Design Phase	In-Game Architecture Design	2025-08-01	2025-08-20
	Game World & Asset Design	2025-08-15	2025-09-15
	UI/UX Wireframing	2025-09-01	2025-09-30
FYP 2: Development	Environment Setup (Godot 4)	2025-10-01	2025-10-15
	Core Mechanics (Movement/Quest)	2025-10-15	2025-11-15
	Compiler UI Implementation	2025-11-01	2025-11-30
	Narrative & Level Design	2025-11-15	2025-12-15
	Build & Optimization (Web/Win)	2025-12-15	2025-12-31
Testing & Conclusion	Functional Testing (Debugging)	2026-01-01	2026-01-10
	User Acceptance Test (UAT)	2026-01-01	2026-01-10
	Data Analysis & Reporting	2026-01-20	2026-01-27
	Final Submission (Milestone)	**2026-01-29**	**2026-01-29**

APPENDIX B: SURVEY QUESTIONNAIRE



Section 1: About You

Just a few quick questions to get to know our players.

How old are you? *

- ◯ Below 19
- ◯ 19 - 20
- ◯ 21 - 22
- ◯ 23 - 24
- ◯ 25 and above

Before playing this game, how would you rate your C++ skills? *

	1	2	3	4	5	
I knew nothing (Complete Beginner)	◯	◯	◯	◯	◯	I was already a pro (Expert)

How often do you usually play video games? *

- ◯ Every day
- ◯ A few times a week
- ◯ Once in a while
- ◯ Almost never

Back Next Page 2 of 6 Clear form

Section 2: The Gameplay Experience

How did the game feel to play?

The game controls (moving and interacting) were easy to get used to. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I could easily understand the UI (Quest Log, Dialogue Box, Compiler Window) *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The game ran smoothly without any major bugs stopping me. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

How do you feel about the visual style of the game? *

1	2	3	4	5
☆	☆	☆	☆	☆

Back Next Page 3 of 6 Clear form

Section 3: Learning & Coding

Did the game actually help you understand C++ better?

The interactive puzzle in game shows a better demonstration than just reading it from book. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The dialogue with characters (like Isyraq, Akari, Zul) explained the concepts clearly. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I feel more confident about basic C++ topics (Variables, Loops, Conditionals, Functions) now. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I would prefer playing a game like this over just listening to a lecture. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Back Next Page 4 of 6 Clear form

Section 4: The Story & Fun Factor

Was the adventure interesting?

The narrative story driven concept made me want to finish the game. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The atmosphere (music, glitch effects, lighting) made the game feel exciting. *

1	2	3	4	5
○	○	○	○	○

The difficulty felt "just right". Not too hard, not too easy. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Back Next Page 5 of 6 Clear form

Section 5: Final Thoughts

Your honest feedback helps me improve!

What was your favorite World? *

- ◯ Dark Woords (World 1)
- ◯ Zen Garden (World 2)
- ◯ Hollowed Core (World 3)
- ◯ Horizon's End (World 4)

Was there any part of the coding puzzles that felt too confusing? *

- ◯ No
- ◯ Other: _____

If you could add/change something to make the game better, what would it be? **(Optional to answer)**

Your answer

Back | Submit | Page 6 of 6 | Clear form