FINANCE JOURNEY IN UNIVERSITY

BY

DICKSON SHEE WEI HAU

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I am extremely thankful and appreciate to my supervisor, Mr. Luke Lee Chee Chien who had given me the opportunity to develop in an IC design project. It was my pleasure to pursue my dream in game development field. Thousands of words were hard to express my thoughts to thank you.

I would like to extend my sincere thanks to my parents for supporting me in everything I like to pursue and always treat me as best as they could. Special thanks to my beloved, Chan Shiao Xuen for always being patient with me, supporting me to pursue my dream and show understanding when I am busy in developing this project.

ABSTRACT

The *Finance Journey in University* project was an educational game designed to improve financial literacy among university students by providing engaging and interactive experience. This game focused on teaching essential financial concepts such as Dollar-Cost Averaging (DCA), budgeting, saving, and investing all within the context of student life. The game presents players with real-world financial challenges including managing tuition fees, living expenses and handling unexpected events such as medical emergencies, family obligations and part-time jobs. By simulating these scenarios, students were encouraged to make informed decisions regarding their finances while maintaining a balance between personal health, mental health and wealth. The project aimed to address the widespread issue of low financial literacy which was prevalent among young adults by providing a practical and hands-on learning tool. Game engine called Unity was used to develop this 2D pixel art open-world game that allowed for the creation of a virtual environment modelled. Players will navigate through various scenes including dormitories, classrooms, entertainment areas and workplaces by simulating the financial decisions that students face daily. The game's innovation lied in its ability to integrate financial education with personal well-being which illustrated the interconnectedness of financial, physical and emotional health. Through this approach, the game encouraged long-term financial planning and decision-making for preparing students to manage their finances effectively in real life. This project also contributed to educational tools that could be implemented in university curricula or financial literacy programs to provide students with the skills necessary for future financial stability.

Area of Study (Minimum 1 and Maximum 2): Game Development, Finance

Keywords (Minimum 5 and Maximum 10): Game Application, Unity, Financial Literacy, University, C#, Pixel Art, 2D Top-Down, Educational Game, Simulation Game, Saving & Investment

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$	US Dollar
%	Percentage
&	And
/	Or

LIST OF ABBREVIATIONS

ACNH	Animal Crossing: New Horizons
LSL	Linden Scripting Language
HUD	Heads-up Display
DCA	Dollar-Cost Averaging
SSAO	Screen Space Ambient Occlusion
MMORPG	Massively Multiplayer Online Role-Playing Game
LTS	Long Term Support
NPC	Non-Playable Character
PC	Personal Computer

Chapter 1

Introduction

This chapter presented the background, motivation and significance of the research while summarising the thesis structure and objectives. It discussed the issues with university students' financial literacy and emphasised the demand for innovative solutions.

1.1 Problem Statement and Motivation

The motivation linked between financial literacy, student behaviour, academic success and long-term financial stability. Financial literacy was not only defined as knowledge but also the confidence and ability to apply that knowledge effectively in real-life financial decisions [2]. University students often encountered with financial difficulties due to limited income opportunities and rising education costs. The cost of undergraduate tuition, room and board in the United States had increased by 42% and 31% at public institutions and private institutions after inflation [10]. As a result, many students were relied on borrowing which contributed to debt accumulation and difficulties in budgeting, saving and making informed financial decisions skills that were crucial for long-term stability. Behavioural economics also presented that emotions often affected rational planning when it came to spending, saving or borrowing which making effective financial management even more challenging for students [11]. Research further showed that young adults aged 18 to 24 possessed the lowest levels of financial literacy among all consumer groups [13]. This greatly overlapped significantly with the university population, leaving students vulnerable to bad financial behaviour, debt and stress. Therefore, closing this gap was important to give students with the tools needed to build financial independence and secure long-term financial well-being [15].

Many university and college students struggled with managing their income effectively. According to research [8], when students had a surplus of money, they often spend it impulsively on both necessary and unnecessary items which reflected a widespread lack of financial literacy. This poor financial behaviour could lead to serious consequences such as excessive debt, risky investments and insufficient savings for the future [11]. Moreover, students often relied on informal social networks and made unrealistic spending decisions which increased their financial burden. Financial stress was a significant issue among students not only due to the inability to meet financial obligations but also due to psychological and emotional impacts [20]. Research showed that students experiencing financial stress were more likely to suffer from anxiety, depression, lower academic performance, and even considered dropping out [20][23]. Financial anxiety was more common among young consumers, especially those who were heavily indebted by using credit card [22].

The goal of *Finance Journey in University* was to create a video game that would give university and college students who might not be very financially literate by providing a foundation in financial literacy knowledge. In the current educational system, practical life skills like money management were often overlooked in favour of academic success. Without guidance, students may not be able to make suitable financial decisions which could have impact on their general well-being. Students will learn through gameplay the value of budgeting, investing and saving while struggling from other aspects of life like academics, mental health and physical health. While there were existing games like *Monopoly* that involve money management but they often lack depth required to teach practical financial skills. This game will also simulate university life and equip students with essential financial knowledge such as Dollar-Cost Averaging (DCA), compound interest and so on [24]. The game not only educated students but also helped to reduce financial stress and supports their journey towards financial independence and security.

1.2 Objectives

This project was intended to create an immersive 2D top-down financial-related game with a pixel art style for PC called *Finance Journey in University* to achieve the following objectives. The primary project's objective was to create an interactive and realistic 2D educational game that would help college and university students become more financially literate. In university life, young adults were often managing money on their own but lacked financial knowledge and abilities required to make wise financial choices. By simulating the day-to-day experiences of a student across a semester, the game aimed to provide players with a realistic immersive environment where they could learn, practise and apply financial principles in an easy way without much effort.

Second, the objective was also to bring out essential concepts of personal finance such as budgeting, saving, investing, Dollar-Cost Averaging (DCA) and compound interest into the project that applicable to students' everyday lives. In order to encourage players to think critically about the consequences of their actions and the significance of long-term financial planning, the project aimed to incorporate these topics into decision-making scenarios and challenges. This approach allowed students to recognise how were their daily choices such as overspending, working part-time or investing small amounts consistently. It could have influenced their future financial stability and opportunities.

Lastly, an important objective was to motivate students to connect financial literacy with their personal aspirations. By demonstrating how effective wealth management could have supported dreams such as purchasing a dream car, starting a business, or travelling abroad, the game aimed to show players that financial planning was not a difficult skill but a foundation for achieving life goals. Lastly, the project aspired to achieve both engagement and education purpose which ensuring that the gameplay experience was enjoyable. It should also educated young adults with valuable financial knowledge at a young stage in their lives.

1.3 Project Scope

The project *Finance Journey in University* was developed as an educational tool and a simulation game that designed to improve students' financial literacy by providing an engaging gameplay experience. The project was presented as a 2D top-down pixel art game built in Unity for PC platform such as Mac and Windows. There were many students faced difficulties in finance caused by challenges like rising living costs, limited financial education and increasing financial pressures.

The scope of the game covered the simulation of a university student's life across one academic semester where players had to navigate everyday financial challenges. Players began with a limited income or allowance each month which had to manage their wealth carefully across those recurring expenses such as rent, transportation, food and beverages alongside unexpected emergencies costs. In addition, the game also included core survival and lifestyle needs such as hunger and thirst, requiring players to spend money regularly on meals and drinks. These game mechanics created a more realistic experience and emphasised the importance of balancing living costs with long-term financial planning.

In addition, the game elements such as status bars for player's health, mental health, hunger and thirst were affected by player's decisions to reflect the student life in realistic. Players had to manage not only their finances but also their behaviour to strengthen the idea that financial decisions were related with quality of life. The core financial concepts such as budgeting, saving, Dollar-Cost Averaging (DCA) and compound interest were applied into the project to encourage players by considering the consequences of overspending and important of consistent investing.

The scope of this project also integrated realistic aspects of student life such as part-time jobs, examinations, and achievements that rewarded good behaviour while the primary focus of this project was to promote financial literacy. Every decision made by player resulted in outcomes that shaped their financial stability, academic performance and personal well-being thereby highlighting the long-term benefits of money management. The project did not applied technologies like advanced artificial intelligence, multiplayer gameplay or professional financial certification modules. The project was delivered as a single-player prototype with core systems and mechanics targeted towards financial education. On the other hand, this project intended to provide students with the knowledge, confidence, and skills to make financial decisions effectively, enabling them to apply these lessons to their real lives and pursue personal goals with financial awareness.

1.4 Contributions

Low financial literacy among university and college students caused long-term financial instability which often led to poor spending habits, rising debt and financial stress. Existing financial education was often theoretical and lack of engagement. Therefore, young adults were failed to prepare for real life practical challenges such as rising education costs and unexpected financial shocks. The COVID-19 pandemic further highlighted the importance of financial preparedness especially among students with limited income sources [3]. This project contributed to addressing the problem between traditional financial education and practical financial literacy by offering a gamified approach. While existing financial learning methods often remained theoretical, *Finance Journey in University* demonstrated how essential that games could be used as an engaging and effective tool to support learning. From an educational perspective, the project contributed by embedding financial concepts such as budgeting, saving, Dollar-Cost Averaging (DCA) and compound interest into interactive gameplay. Unlike conventional teaching which often presented these ideas directly, the project enabled students to apply the skills directly in simulated real-life scenarios by enhancing long-term financial awareness and decision-making skills.

In gameplay perspective, the project contributed by combining financial education with life simulation mechanics such as hunger, thirst, health and mental health. This created a more realistic and relatable environment compared to traditional board games or simplified financial apps. It helped students to understand the differences between essential needs and long-term goals.

Finally, this project contributed practically by equipping young adults with a stronger foundation in personal finance during young age in their lives. The project motivated players to view financial literacy as a pathway towards achieving personal goals such as entrepreneurship, travel or purchasing a dream car. All in all, these contributions highlighted the potential of engaging games as an innovative tool for improving financial literacy among university and college students. It was suitable for educational purposes by making a game related to sector which might attract students to dig out more knowledge.

1.5 Report Organization

The structure of this report was organised as follows to provide a comprehensive overview of the development process and research behind the financial literacy educational game, *Finance Journey in University*. Chapter 1 was the Introduction which presented the introduction of this project by including several sub-topics. Chapter 2 which was Literature Review, reviewed related technologies and existing applications by analysing their mechanics, educational effectiveness, strengths and limitations. System Methodology and Approach as Chapter 3 outlined the overall design framework and proposed method for the game. Chapter 4 described the System Design of the project by providing top-down system design diagrams and explaining how each functionality was created for the project. Chapter 5 was System Implementation to discuss the setup of the project with specific settings and performance evaluation. Chapter 6 represented System Evaluation and Discussion which discussing system testing including how it was set up and carried out alongside with project challenges, objective evaluation and a concluding remark. Finally, Chapter 7 concluded the entire project and presented recommendations for future improvements and further development.

Chapter 2
Literature Review

2.1 Review of the Technologies

This section discussed game development technologies about the game engines which helped developers in implementing a game application and compared them in several aspects. A game engine was defined as a software framework designed to simplify the development of video games by integrating essential components such as rendering, animation, audio, artificial intelligence, networking and game mechanics into a unified platform. For this project, three popular game engines, namely Unity, Unreal Engine and Godot, were reviewed that suitable for 2D game development [1].

2.1.1 Unity

Unity used C# as its scripting language at the beginning which having phased out Boo and JavaScript in earlier versions of the engine [19]. The engine adopted component-based architecture that game objects were assembled through modular components rather than only relying on strict inheritance hierarchies. This design allowed developers to increase flexibility in development workflows [1]. Since it was initially released in 2005 for Mac OS, Unity expanded to support a wide range of platforms which include Windows, Linux, macOS, mobile systems, gaming consoles and virtual reality devices such as Oculus Rift and PlayStation VR [1]. In addition, Unity integrated with multiple graphics APIs including Direct3D, OpenGL, OpenGL ES, WebGL and console-specific APIs [19]. It provided dedicated tools for both 2D and 3D development by supporting features such as sprite-based animation, terrain editing, reflection mapping, texture compression and advanced rendering techniques such as screen space ambient occlusion (SSAO) [19].

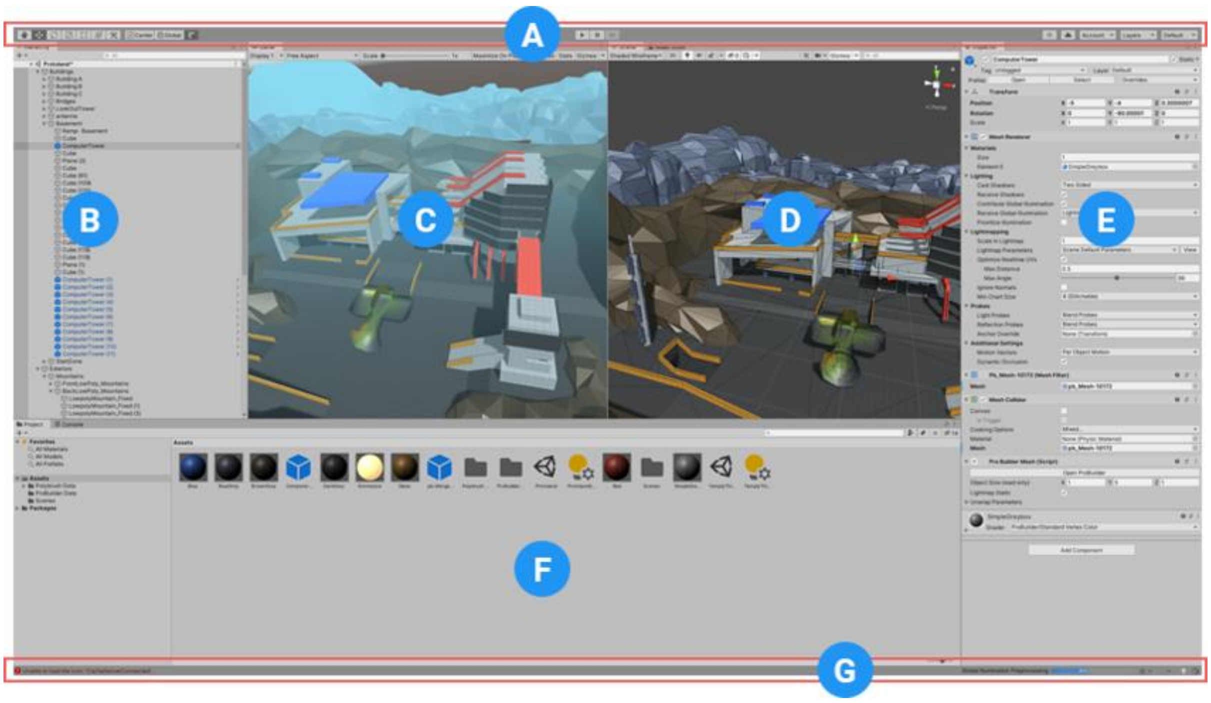


Figure 2-1-1 Unity Interface [26]

Unity provided 2D game development packages with features like built-in physics for 2D environments, skeletal animation for 2D characters, tilemap support, and sprite editors. These tools made level design, character animation and asset management more efficient in game development aspect. Furthermore, Unity's interface and workflows for 2D projects were reviewed as intuitive and beginner-friendly by enabling new developers to get familiar with the basics of game design without facing steep technical barriers. The asset store in unity provided extensive tutorials and active developer community which reflected that Unity was widely considered as one of the easiest engines for developing 2D games especially for independent developers and students [1].

In terms of pricing, Unity was available in different tiers depending on revenue of established game. The free Personal version provided most features but was restricted to developers with annual earnings under $100,000. Developers that making higher revenue could apply for Plus or Pro version which included advanced services such as enhanced analytics, performance and bug reporting, and dedicated technical support [1].

Unity was also widely supported in the game industry with a broader adoption compared to other engines such as GameMaker Studio 2 and Cocos2d-x. Its cross-platform portability ensured the popularity to reach a wider audience across various devices [9].

2.1.2 Unreal Engine

Epic Games created a game engine called Unreal Engine that was originally created in 1998. It was initially designed for first-person shooters then later expanded to support additional genres such as MMORPGs, stealth and adventure games which demonstrated its flexibility within the gaming industry [19]. It was written in C++ programming language and provided strong portability by enabling deployment across multiple platforms including Windows, macOS, Linux, iOS, PlayStation, Xbox, Android, Nintendo and virtual reality systems such as Oculus Rift, HTC Vive and SteamVR [1][19].

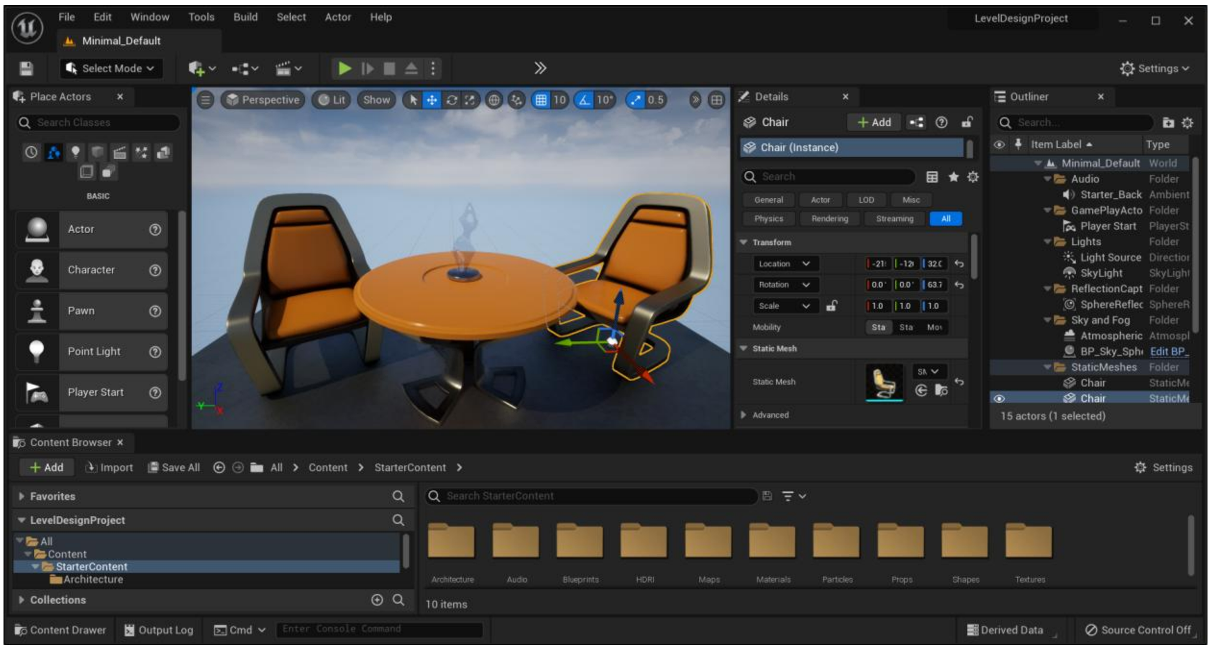


Figure 2-1-2 Unreal Engine Interface [25]

A key feature of Unreal Engine was the Blueprint visual scripting system. It allowed developers to design game logic through graphical diagrams provided in game engine. This feature could solve the problem of non-programming background staff required to write write C++ code and made the engine accessible to both programmers and non-programmers [1]. The engine was also equipped with a range of built-in modules including PhysX for physics simulation, DirectX/OpenGL/WebGL for graphics, a sound engine, artificial intelligence systems, post-processing effects and networking functionalities which supporting advanced game development needs [1].

For 2D development, Unreal Engine provided functionality but less considered about intuitive than Unity. 2D projects could be created in Unreal Engine but the workflows and tools were more complex which made adoption process more difficult for beginners. Unity was generally regarded as superior for dedicated 2D development due to its dedicated 2D mode whereas

Unreal's strength laid primarily in 3D environments. Nevertheless, Unreal supported 2D game development through features such as sprite rendering and physics which making it possible to build 2D games with a steeper learning curve [1].

In terms of licensing, Unreal Engine was free to use with full features provided that a game's revenue did not exceed $3,000 per quarter for game developers. 5% royalty was payable to Epic Games if the game earnings surpassed the threshold [1]. The engine was also received multiple awards such as recognition by Guinness World Records as one of the most successful game engines [19].

2.1.3 Godot

Godot was an open-source and free game engine which supported 2D and 3D game application development which was provided in multiple platforms like desktop, mobile and web [12]. It was particularly popular for 2D projects because of its lightweight design, dedicated 2D engine that was not dependent on 3D components, and an intuitive scene system that simplified the organisation of game elements. This made it highly suitable for beginners and small teams focusing on 2D development.

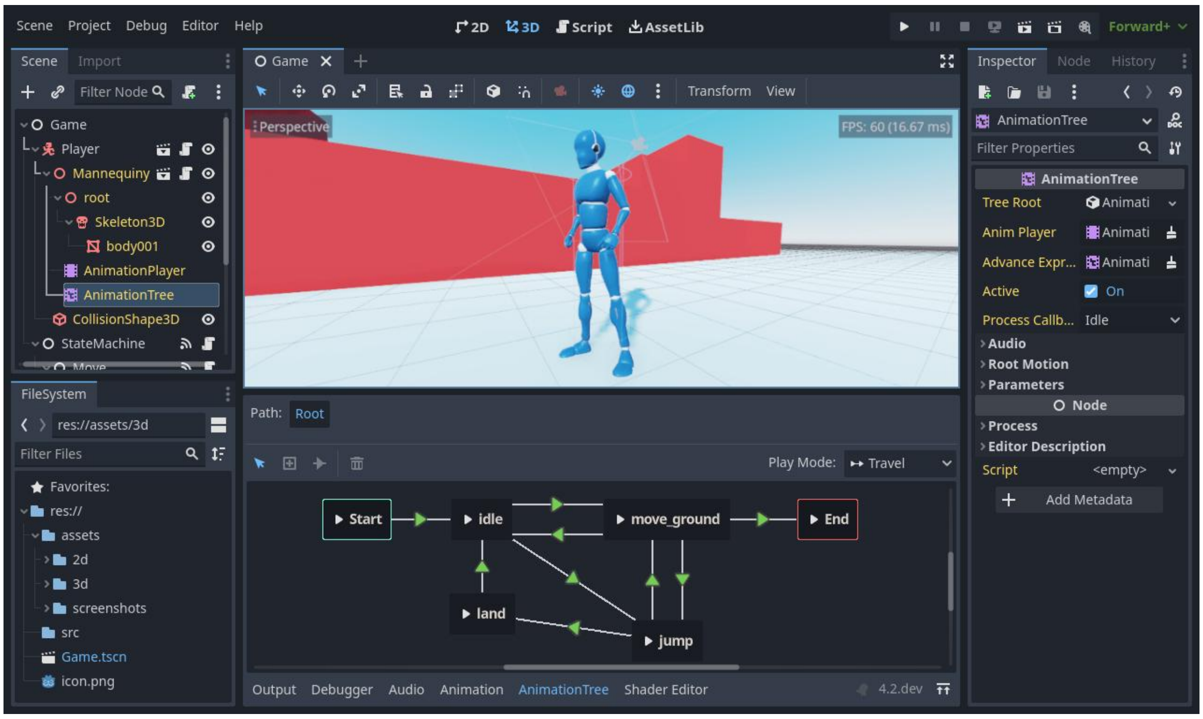


Figure 2-1-3 – Godot Interface [12]

Godot provided a user-friendly approach to learning by supporting GDScript which was a Python-like language designed especially for the engine. It was easy for developers who were already familiar with Unity as it supported C#, lightweight and simple to understand. With

GDExtension, developers could write performance-heavy code in C or C++ while still integrating smoothly with the engine [12]. These multiple programming language options gave learners flexibility to develop game application depending on their background and made the engine accessible to both beginners and experienced developers.

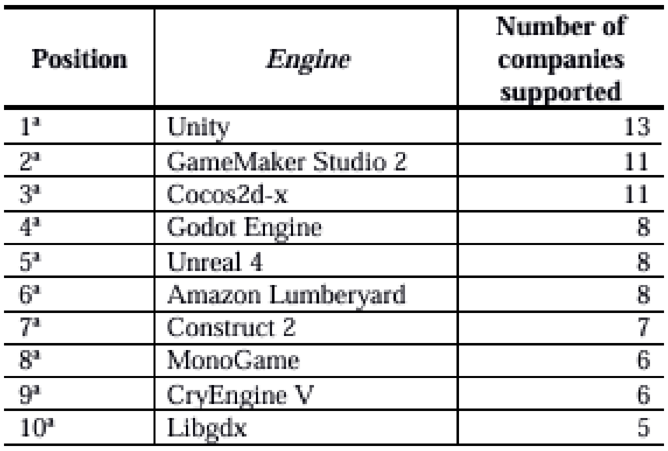
Additionally, Godot supported programming languages including C# 7, C++ and GDScript which provided developers with variety of options depending on the game development requirements [1]. Godot was entirely free and open source, it attracted independent developers and small studios that looking for a powerful 2D game engine.

2.1.4 Comparison Between Game Engines

When comparing game engine like Unity, Godot and Unreal Engine for this project, several important factors were compared. In terms of learning curve and accessibility, Unity was widely recognised as more beginner-friendly because of its structured documentation, intuitive interface and use of C# which was easier to adopt compared with lower-level languages such as C++ [1][19]. This was reinforced by Unity's collection of tutorials and training materials that were designed not only for independent learners but also for academic settings. This strength made it suitable for students who aimed to apply these skills in both educational and professional contexts [9]. Godot also offered accessibility with its scripting language GDScript which resembled Python and allowed beginner to pick up programming quickly. However, its educational resources and community content were not as extensive as Unity's [12]. Unreal Engine was known for its professional-grade tools but also came with a steeper learning curve especially due to its reliance on C++ as the primary language. Although it offered Blueprints visual scripting system that helped non-programmers build logic without coding but mastering Unreal still required more technical effort compared to Unity or Godot [1].

For 2D game development, Unity gained a clear advantage. It offered 2D game development packages with optimised tools for sprites, tilemaps, animations and physics that supported by an extensive ecosystem of pre-built assets and plugins from the Unity Asset Store. Godot also provided a strong 2D engine with a lightweight and node-based system that was intuitive for structuring 2D scenes [12]. However, its relatively smaller ecosystem meant that developers might need to create more features from scratch. Unreal Engine was heavily optimised for high-end 3D graphics. Although it provided support for 2D game development via Paper2D, the feature had not been updated or adopted by the community to the same extent as the tools offered by Unity or Godot [1]. The research showed that Unreal was less suitable for 2D-focused educational games where efficiency and streamlined workflows were more critical than cutting-edge graphics.

Furthermore, another important factor was the size and activity of each engine’s community and ecosystem. Unity benefited from one of the largest developer communities which offering abundant resources such as forums, tutorials, third-party courses and an active Asset Store where developers could access to reusable code, assets and tools. Despite having a smaller ecosystem but Godot's community was enthusiastic and rapidly expanding due to its open-source characteristic. Users regularly contributed tutorials, plugins, and improvements in Godot’s community as well [12]. Unreal Engine also boasted a strong professional community with extensive documentation, marketplace content and industry recognition. However, its ecosystem was more focused on high-end 3D projects which meaning smaller scale 2D educational projects benefited less directly from its strengths [1].



Position	***Engine***	**Number of companies supported**
1ª	Unity	13
2ª	GameMaker Studio 2	11
3ª	Cocos2d-x	11
4ª	Godot Engine	8
5ª	Unreal 4	8
6ª	Amazon Lumberyard	8
7ª	Construct 2	7
8ª	MonoGame	6
9ª	CryEngine V	6
10ª	Libgdx	5

Figure 2-1-4 Ranking of the engines according to their support to the companies [9]

Based on Figure 2-1-4, Unity demonstrated the highest level of company support, with 13 major companies such as Google, Apple and Facebook backing the engine [9]. In comparison, Unreal Engine had the support of eight companies, including Microsoft, Ubuntu and Nintendo, while Godot Engine was supported by eight companies, such as Oculus, Valve and Microsoft. This research demonstrated that Unity provided broader industry support and greater opportunities which enhanced its portability across different platforms. The wider the industry adoption of a game engine, the stronger its potential to reach larger audiences.

In summary, the comparison of the game engines indicated that Unity was the best option for this project because it provided strong, efficient tools for creating 2D games. Godot’s environment was still developing, it turned out to be better choice for smaller project. On the other hand, Unreal Engine was less suitable for 2D-focused educational games due to its complexity and more demanding development requirements.

2.2 Review of the Existing System/Application

2.2.1 Animal Crossing: New Horizons (ACNH)

Animal Crossing: New Horizons (ACNH), released in 2020 for the Nintendo Switch, was an open-ended game that fell under the adventure, puzzle, and life simulation genres [5]. It was released during the height of the first COVID-19 lockdown in March 2020 and became an instant success, generating $654 million in revenue and ranking as the seventh top-grossing premium game of 2020. Zhu [5] delved into the game's psychology, suggesting that it offered players a temporary escape from harsh realities, though there was a risk of excessive escapism. However, the virtual environment could also serve as a valuable social platform, enabling players to maintain social connections and alleviate loneliness. ACNH was developed using Nintendo's proprietary game engine, optimized for the Nintendo Switch console. The engine allowed for the seamless integration of real-time, real-world synchronization where in-game time corresponds to the actual time. Furthermore, the game featured a stylized and cartoonish art direction with a focus on charming and colourful aesthetics. The character models, environments, and animations were designed to be appealing and accessible to a broad audience, contributing to the game's relaxing atmosphere.

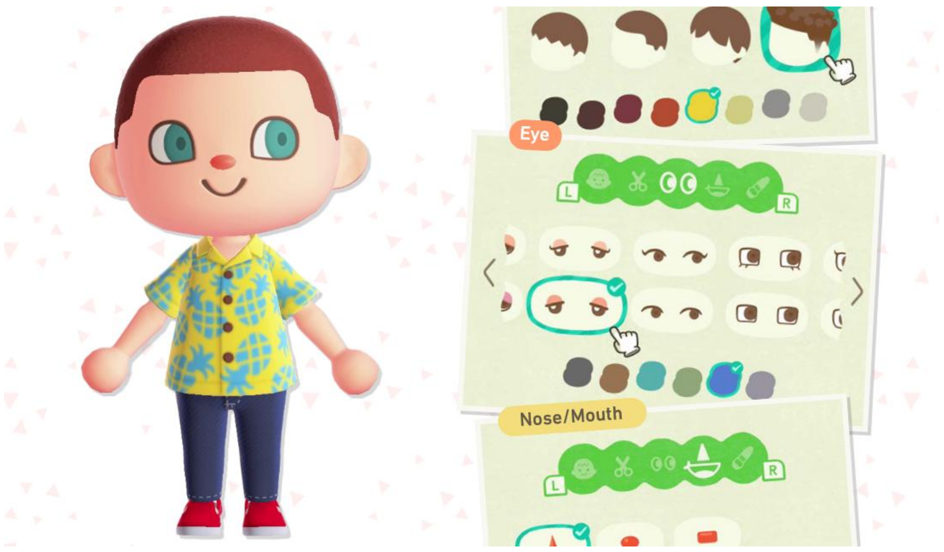


Figure 2-2-1 Customization of Character in ACNH

First, players began their journey on a deserted island inhabited by animal characters and were free to set their own goals to complete various activities [17]. Based on Figure 2-2-1, players could customize their character when creating a new character in a new world. Based on several studies [16], it had been found that the ability to customize a game character whether in appearance or abilities was a strong predictor of player identification with that character. Researchers explained these findings through the self-discrepancy theory. Specifically, allowing players to design a character that closely resembled their ideal self-help reduced the gap between their actual and ideal selves, increasing the likelihood that they will identify with the character. It acted as a factor that attracts players to play ACNH which offered high freedom to set up a character they like. Apart from that, players could interact with the island's inhabitants, fish, plant trees, collect bugs, design and build homes. The game also offered a creative platform where players could craft items such as tools, clothing, and furniture from collected resources as well as decorate their island and homes.

Figure 2-2-2 Tom Nook, a raccoon dog in ACNH [17]

Soon after arriving on the island, players quickly realized that their adventure was not a simple vacation. The island was managed by Tom Nook in Figure 2-2-2, a friendly yet ambitious capitalistic property developer. Tom, a raccoon dog (Tanuki), informed the new residents through text-bubble monologues that they were in debt to him and his company, Nook Inc. for the cost of their relocation. To settle the debt, residents were provided with a tent, a cot and a smartphone and were instructed to spend their time gathering island resources such as weeds, sticks and plants to repay their debt and enhance their island experience [17]. Players needed to interact with Nook Inc. employees and occasional visiting sales-animals to buy and sell items using the island's currency, known as "Bells". Additionally, they could access various items and upgrade through the Nook Miles loyalty program, which was available on the smartphones issued by Nook Inc. ACNH fulfilled all needs in Maslow's Hierarchy of Needs as examined by researchers [4]. They assessed how engagement in the virtual world of ACNH could help satisfy real-world needs and consider the needs of the in-game avatars and villagers who existed solely in the virtual environment to explore how their needs were met.

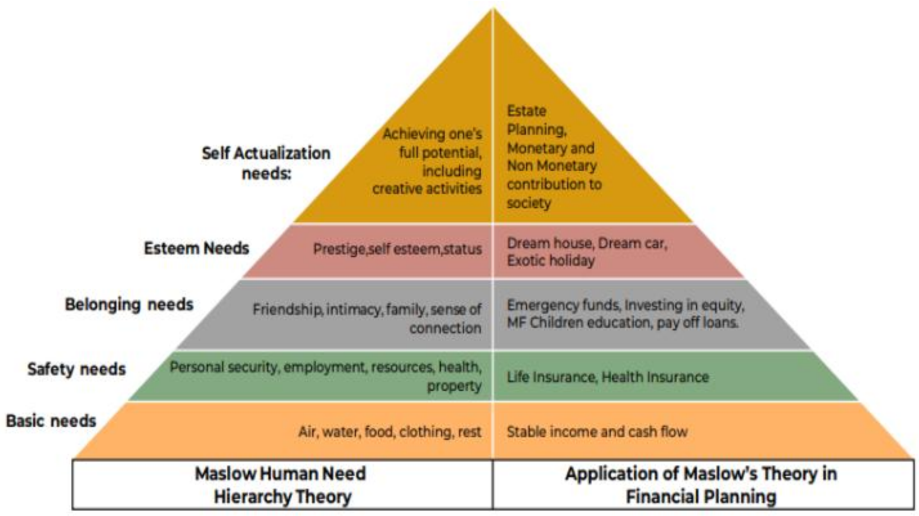

Self Actualization needs:
Achieving one's full potential, including creative activities
Estate Planning, Monetary and Non Monetary contribution to society
Esteem Needs
Prestige,self esteem,status
Dream house, Dream car, Exotic holiday
Belonging needs
Friendship, intimacy, family, sense of connection
Emergency funds, Investing in equity, MF Children education, pay off loans.
Safety needs
Personal security, employment, resources, health, property
Life Insurance, Health Insurance
Basic needs
Air, water, food, clothing, rest
Stable income and cash flow
Maslow Human Need Hierarchy Theory
Application of Maslow's Theory in Financial Planning

Figure 2-2-3 Financial Planning Through Maslow's Human Need Theory [18]

In human psychology, Abraham Maslow introduced a theory called Maslow's hierarchy of needs in 1943 [18]. Abraham Maslow explained that as we fulfilled our basic needs, we progress to higher levels of behavioural development. Once our fundamental needs such as food, clothing, and shelter were met, we moved on to safety needs. After that, we strived for higher levels of the hierarchy such as self-actualization and social esteem. This concept could also be applied to financial planning based on Figure 2-2-3. Human needs were prioritized starting with the basic survival aspect of wealth creation, eventually leading to the highest level which involved wealth distribution for self-actualization. It was important for our spending and investments to align with these levels to ensure a balanced and stable financial life. For example, if someone prioritized esteem needs over belonging needs, they might accumulate costly debts and reduce the money that could otherwise be invested in growth opportunities like equities or mutual funds [18].

Animal Crossing: New Horizons (ACNH) could influence players' financial literacy by simulating various aspects of money management and resource allocation in a virtual environment. Although the game does not directly teach financial concepts, it offered opportunities for players to practice financial decision-making. Players earn in-game currency, "Bells," which they could use to trade items, manage loans for home upgrades and invest in resources like turnips in the "stalk market" in the game which was a simplified version of real-world stock trading. The stalk market showed different prices for items at different times by simulating real-world stock trading and teaching players financial knowledge about stock trading. Through these activities, players indirectly learn about budgeting, saving and the consequences of financial decisions such as managing debt, balancing spending on consumable items versus investments and understanding the value of assets over time. This practical exposure to managing resources could subtly enhance players' financial awareness and decision-making skills which offering a basic understanding of economic principles like supply and demand, investment strategies and cost management in a non-threatening and engaging way.

In conclusion, the connection between gaming and financial literacy was discussed in various articles, exploring how video games simulate real-world financial behaviours and affect decision-making skills. Research showed that games like ACNH help players developed better financial habits by encouraging resource planning and money management in a low-risk environment. However, the game introduced basic financial concepts like debt and currency management but does so in a highly simplified manner. Real-world financial complexities, such as interest rates, credit scores, and long-term investment strategies, were not addressed.

2.2.2 Monopoly Board Games

The board game *Monopoly* originally began as an educational tool intended to critique monopolies. Its inventor, Elizabeth Magie patented the game in 1903 under the name *The Landlord's Game*, a 19th-century economist and political figure [27]. In 1989, Knechel introduced *Monopoly* into undergraduate accounting education to help students learned how to make financial accounting journal entries [20]. This method was designed as an alternative to the traditional cumulative practice sets commonly used in introductory accounting courses which often failed to engage students that lacked connection to real-world business practices and posed grading difficulties due to identical solutions across all students. To foster active participation and a clearer grasp of debits and credits, one of the authors adopted Knechel's modified *Monopoly* approach for use in principles of accounting course.

Through the gameplay of *Monopoly*, players intended to buy, rent and sell properties to increase their wealth and become the wealthiest player. Starting at the "GO" space, a player could buy a property from the bank, or it gets auctioned to the highest bidder. Players who owned properties collect rent from others who land on those spaces and the construction of houses and hotels increases rent. Players also deal with taxes, Chance and Community Chest cards and may occasionally be sent to jail. The objective was to be the last player remaining who was not bankrupt [7].

The *Monopoly* accounting game was introduced by Ohio State University as a learning medium in 1963 [14]. This highlighted the potential advantages of using game-based media for teaching financial accounting which allowed students to tackle unique problems compared to their peers. Albert [7] proposed that the *Monopoly* accounting game served as a simulation that generated transactions for students to practice the accounting process where each transaction was recorded and summarized into a financial report. The *Monopoly* accounting game acted as a learning medium to bring more interesting learning process to students so that the learning process could run optimally. On the other hand, the advantage of playing *Monopoly City*™ with students was that it created clear connections between their prior understanding of how the property market worked through gameplay and the new theoretical concepts taught in the course. By making these connections, students could quickly apply these new concepts to their existing knowledge of the real-world property industry which accelerated their understanding and adaptation. As a result, they grasped the material more quickly as they progress through their studies [7].

In conclusion, the *Monopoly* board game had brought many advantages related to financial literacy by helping players developed a better understanding of various financial concepts like budgeting, money management, investment, debt management and more. Players managed in-game currency ("Monopoly money") and learned to balance spending on properties, taxes and fines while trying to maintain enough cash for future expenses. In addition, players decided whether to invest in properties, build houses or hotels and participate in auctions. These decisions teach basic investment strategies, risk assessments and long-term financial planning. The game also included mortgages and loans by helping players understand how borrowing works, the impact of interest and the risks associated with debt. By buying properties and collecting rent from other players, participants learned the concept of passive income and the importance of asset ownership in building wealth. Overall, *Monopoly* provided a fun and interactive way to practice financial literacy concepts such as saving, investing, and budgeting while fostering an understanding of financial risk and opportunity.

2.2.3 Financial Literacy Game using Second Life

The project researched called "*Financial Literacy Game using Second Life*". It was a role-playing game designed by researchers at Ohio University in the United States that could be played by one player or by several [6]. Throughout the game, the player assumed the role of an 18-year-old who had just graduated from high school. They must make decisions about when to rent an apartment or when to buy a house which career to pursue whether to attend college or start working right away and whether to invest in retirement accounts.

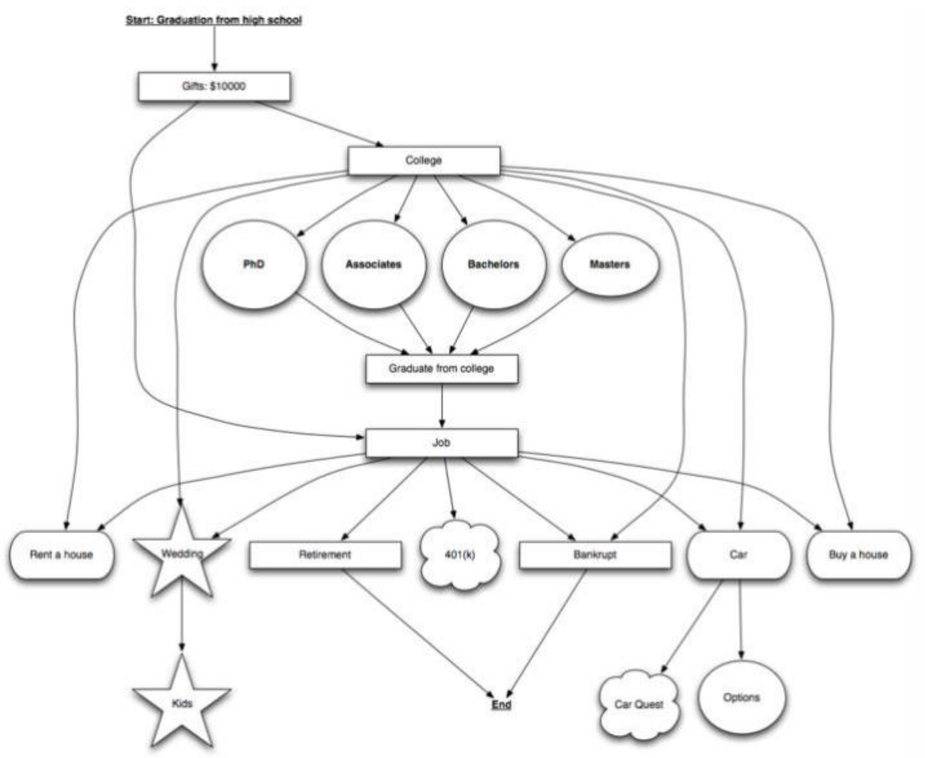
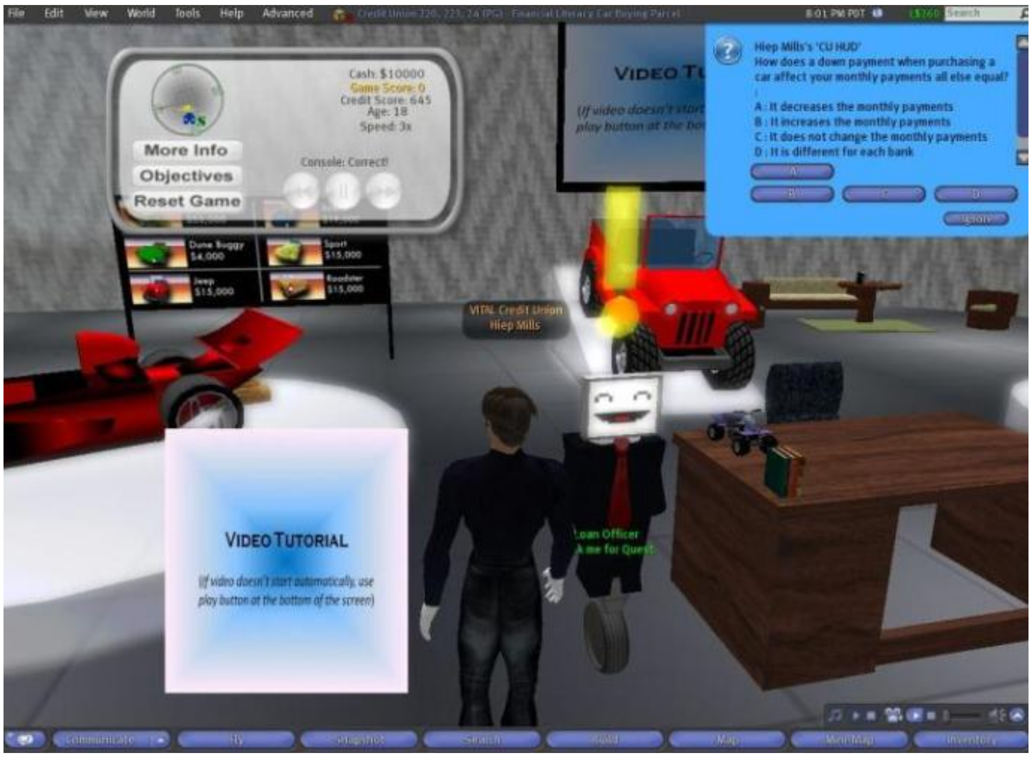


Figure 2-2-4 Game Design Diagram for *Financial Literacy Game* [6]

Figure 2-2-4 presented the overall game design diagram by showing the different paths a player could take. Every few seconds as the game progresses, the player's financial situation was updated to reflect two weeks of their simulated existence. Bankruptcy could result from poor money management before the player reaches the 65-year-old retirement age. Nevertheless, the player's wealth upon retirement will be a major factor in calculating their ultimate game score if they manage to reach it. The expectation that players will fail multiple times before succeeding was another crucial aspect of the game. They were forced to acquire fundamental financial knowledge and skills as they progress through the game toward a fulfilling retirement since it was the only way to prevent early bankruptcy and keep making progress in retirement savings.

The game was created with *Second Life* which made use of the Linden Scripting Language (LSL) [6]. LSL was an event-based language that enabled users to manage the actions of avatars and objects. Players had the ability to generate new functions using a selection of over 300 pre-installed options. LSL scripts were connected to objects to alter how they function. After being written, these scripts were saved and processed by Linden Labs' servers. The game utilized a Heads-up Display (HUD) that players could buy and equip onto their avatar for gameplay. The island featured programmed elements like vehicles, signs advertising sales and financial experts that engaged with players while they're playing. When a player selected an object, it transmitted a notification to the island specifying the avatar involved. HUD read and processed these messages with various scripts. Because *Second Life* had memory constraints, the HUD scripts were split into various logical functions such as managing conversations, controlling events, calculating credit scores and handling quests. Moreover, scripts often interacted to accomplish tasks such as during the process of buying a vehicle. The car script verified if the player had a trade-in and asked for the player's credit score from the credit script. The credit script evaluated the credit score by examining other scripts such as home, cash and age. After being calculated, the score was sent back to the car script to enable the purchase to continue. To enhance performance and minimize lag, scripts were spread out among various objects within the HUD to ensure effective data communication. Scripts that frequently exchanged data were grouped together in the same object to reduce the number of messages being transmitted. There were several quests for players to complete which aimed to improve their level of financial literacy. Players were required to answer a series of questions designed to test their understanding of various financial situations.


Hiep Mills's 'CU HUD'
How does a down payment when purchasing a car affect your monthly payments all else equal?

A : It decreases the monthly payments
B : It increases the monthly payments
C : It does not change the monthly payments
D : It is different for each bank

Cash: $10000
Game Score: 0
Credit Score: 645
Age: 18
Speed: 3x
Console: Correct!

Figure 2-2-5 Car Purchase Option in *Financial Literacy Game* [6]

As shown in Figure 2-2-5, these questions appeared when players made significant life decisions such as buying some property. Before answering, players had the option to watch a video that provided relevant information for the task. These quests were intended to educate players on the important details they should consider when making major financial purchases.

The *Financial Literacy Game* featured quests where players encountered real-life financial scenarios such as purchasing a car or deciding whether to rent or buy a house. During these quests, players watched videos that simulated these situations such as interacting with a car dealer. After viewing the video, players answered approximately four multiple-choice questions to test their financial knowledge on the topic [6].

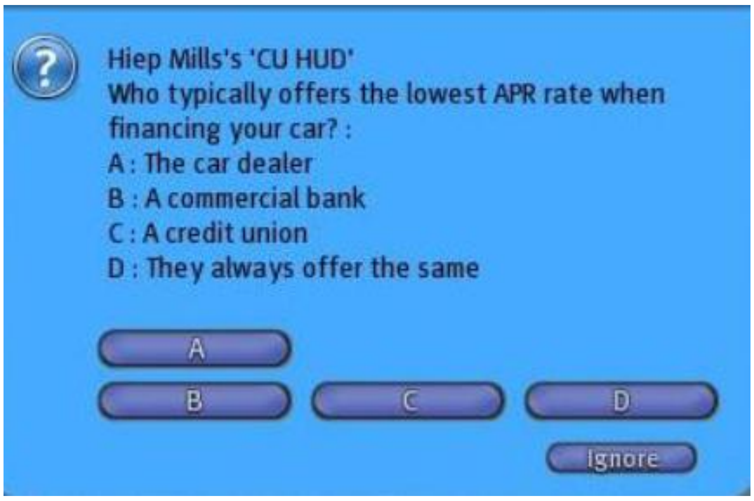
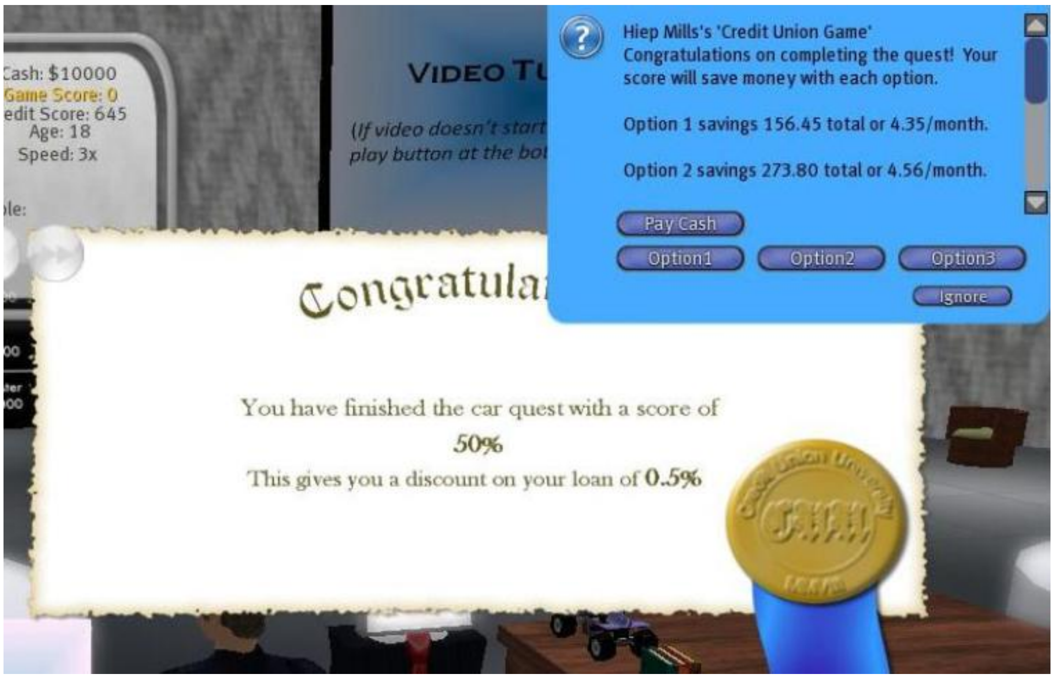

Hiep Mills's 'CU HUD'
Who typically offers the lowest APR rate when financing your car? :
A : The car dealer
B : A commercial bank
C : A credit union
D : They always offer the same

Figure 2-2-6 Car Quest Question [6]


Hiep Mills's 'Credit Union Game'
Congratulations on completing the quest! Your score will save money with each option.

Option 1 savings 156.45 total or 4.35/month.

Option 2 savings 273.80 total or 4.56/month.

Cash: $10000
Game Score: 0
edit Score: 645
Age: 18
Speed: 3x

Congratula

You have finished the car quest with a score of
50%
This gives you a discount on your loan of 0.5%

Figure 2-2-7 Players Receive Car Loans by Answering Questions [6]

By answering questions, correct answers will lead to better deals from the dealer as shown in Figure 2-2-6 and Figure 2-2-7. The quests focused on basic financial information to keep players engaged with plans for future updates to add more detailed and informative content.

The purpose of the *Financial Literacy Game* was for players to recognize the significant costs associated with purchasing some property, managing everyday expenses and to improve players' level of financial literacy [6]. Key topics essential for financial literacy include:

- **Money Management**, which involves learning about budgeting and understanding credit scores, along with their impact on other financial decisions.
- **Financial Planning** involves setting clear financial goals, preparing for retirement, and keeping detailed and accurate financial records.
- **Introduction to Investing** covers the basics of financial tools such as checking and savings accounts, certificates of deposit, and various investment options. It also explores tax-advantaged accounts and retirement plans like IRAs, 401(k), and 403(b).
- **Insurance Awareness** highlights the significance of different types of insurance to protect against financial loss.
- **Housing Choices** help players understand the pros and cons of renting versus buying a home, navigate the decision-making process, and learn about the steps involved in selling and relocating.
- **Career Choices**, helping players decide between attending college or seeking employment, choosing a major, and considering part-time work during school.

2.2.4 Wealth Creation Game for Financial Literacy

The *Wealth Creation* game was a resource management board game designed to teach financial literacy by simulating real-world financial decision-making. Developed by researchers from Kasetsart University in Bangkok [14], the game incorporated key financial concepts like managing loans, investing and budgeting that offered players a hands-on experience of wealth accumulation. Players must navigate various financial risks and unexpected events which made decisions that mirror real-life financial challenges. The primary goal of *Wealth Creation* was for players to accumulate as much wealth as possible through strategic investments and financial management. The game progressed through four rounds by each representing different stages of life. Initially, players spend most of their income on necessities which as the game advances, they could diversify investments, trade assets and pursue higher-yield opportunities. By the end of the game, players could choose to sell risky assets for maximum profit or retain those with higher expected returns. The player with the highest accumulated wealth wins.

Mechanic	**Financial concept**	**Action**
Draw and select	People needed to manage their income by allocating it for spending, investing, saving, and protecting against risks.	During each round, the player selected three out of five cards that provided instructions on income, investments, insurance and wealth management.
Open drafting	People could opt to invest in assets that align with their risk tolerance and available investment budget.	Players could select from a variety of assets in a pool with each asset type having certain restrictions.
Income	Individuals could earn money through various means such as employment and investing.	Players earn income from their investment asset cards and work-related cards.
Loan	Loans could be utilized for various purposes such as covering basic living costs and investing in real estate. Each type of loan came with a different interest rate.	Players could borrow money to participate in events or invest in real estate cards but they must repay the loan's interest by the end of the round.
Stock holding	People could build wealth by investing a portion of their income.	Players will earn returns in the form of dividends, interest or capital gains.
Market	The trading prices of common stocks on stock exchanges were affected by market demand.	When you buy or sell stocks, their value fluctuates. As demand for purchasing increases, stock prices rise, and when demand decreases, prices fall.
Event	Economic conditions and government policies influence both spending and investment.	Players have the option to buy insurance to cover expenses related to illness or property damage.
Protection	Individuals could safeguard their assets from potential loss by purchasing insurance.	Players could buy insurance to secure payouts in the event of illness or property damage.
End game scoring	Wealth was the gathering of valuable economic assets which could be quantified based on physical goods or money.	The worth of money and assets like real estate, common stock, bonds, and gold, will add up as the total score at the last game.

Table 2-2-1 Game Mechanics in *Wealth Creation* [14]

The game mechanics were crafted to align with the financial aspects which enabled players to make decisions within a simulated environment aimed at building financial wealth. Table 2-2-1 provided a detailed description of the design mechanism [14].

Figure 2-2-8 Game Component of *Wealth Creation* [14]

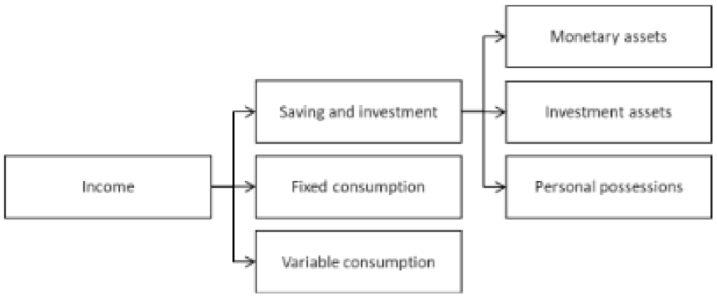


Figure 2-2-9 Content Structure of *Wealth Creation* [14]

Figures 2-2-8 and 2-2-9 illustrated the game components and the financial content structure. *Wealth Creation* integrated roll-and-move mechanics, open drafting, events, income generation, investment opportunities and loan management into the gameplay. Each player started with 30,000 baht and income cards representing full-time, freelance or temporary employment. The game was played across four rounds, each consisting of five phases:

- **Expenditure Phase**: Managing spending decisions
- **Investment Phase**: Acquiring assets
- **Event Phase**: Encountering financial risks or opportunities
- **Income Phase**: Earning money from investments and work

- **Loan Interest Phase**: Repaying any borrowed money

Each phase reflected real-world financial scenarios. The final wealth score was determined by the total value of cash, assets, and goods accumulated by the end of four rounds. The player with the highest wealth wins. The game's mechanics closely mirror financial principles such as credit limits, market demand, and investment returns.

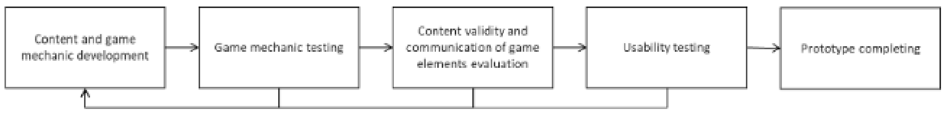

Content and game mechanic development
Game mechanic testing
Content validity and communication of game elements evaluation
Usability testing
Prototype completing

Figure 2-2-10 Development process of *Wealth Creation* [14]

The development of *Wealth Creation* followed a structured process that included four phases, as shown in Figure 2-2-10 [14]:

- **Content and Game Mechanic Development**: Developers researched financial literacy topics and integrated them into game mechanics.
- **Game Mechanic Testing**: Experienced gamers tested the game's flow and balance.
- **Content Validity and Communication**: Financial content was validated by an economics professor, while game designers evaluated the clarity of financial elements.
- **Usability Testing**: Teachers and high school students assessed the game's educational value and suitability for classroom use.

Wealth Creation covered essential financial literacy topics such as money management, investment strategies, loan management and insurance. Players were introduced to budgeting and investing in assets like real estate and stocks and understanding market fluctuations. The game also teached the importance of protection through insurance and risk management. The game's design allowed players to balance short-term spending with long-term investments to make strategic choices based on risk tolerance and manage debt. Players also learned the importance of diversifying their investments and preparing for unexpected financial events like property damage or illness, which affect their overall wealth.

In short, *Wealth Creation* provided an engaging and interactive platform for students to learn financial concepts in a practical and enjoyable way. It reinforced key financial literacy principles such as budgeting, investment diversification, risk management and credit management through real-world simulations. The game demonstrated the effectiveness of game-based learning in enhancing financial literacy education.

2.2.5 Summary of The Existing System

The literature on financial literacy games presented a variety of educational tools, each with distinct strengths and limitations.

Animal Crossing: New Horizons **(ACNH)** offered a relaxed virtual environment where players managed in-game currency and engage with basic financial management concepts. It introduced ideas such as debt repayment and resource allocation in an accessible in enjoyable way. However, the game's financial system was highly simplified which lacked deeper financial principles such as interest rates, credit scores and long-term investment strategies. As Zhu [17] highlighted, another limitation was the risk of encouraging excessive escapism. The immersive and fictional world of ACNH could detach players from real-world financial challenges that were potentially diminishing its educational impact.

Monopoly had also been adapted for financial education, teaching concepts like property investment, debt management and strategic financial decision-making. Its strengths lied in fostering an understanding of budgeting, money management and risk assessment within a competitive environment. Nevertheless, *Monopoly's* limitations included its highly controlled and oversimplified financial environment. Players faced straightforward decisions regarding property ownership and taxes without encountering the unpredictability of real-world financial markets. Furthermore, *Monopoly's* emphasis on competition and wealth accumulation could overshadow important financial literacy lessons such as the importance of long-term planning, diversification and sustainable financial behavior.

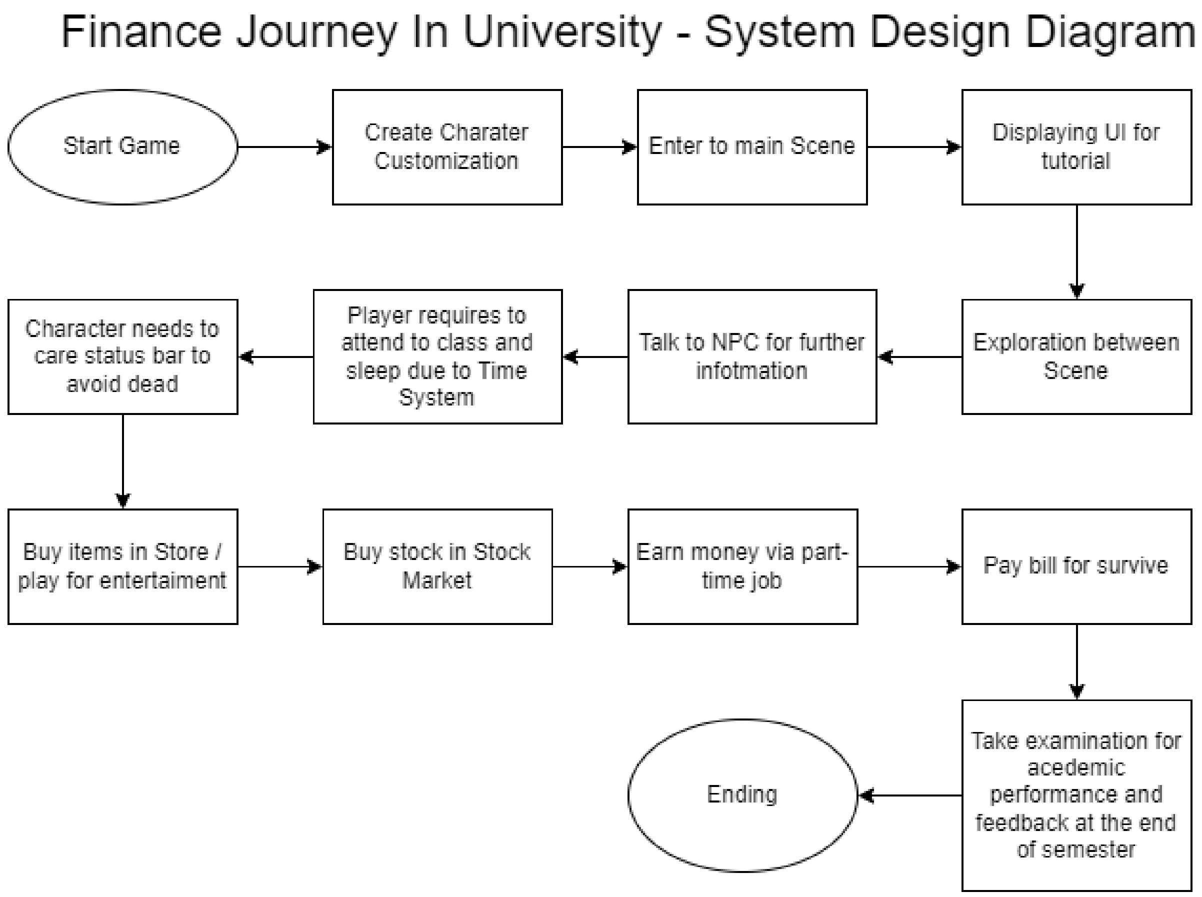
The ***Financial Literacy Game using Second Life*** introduced players to real-life financial scenarios through role-playing simulation. This game provided a rich and interactive platform where users could practice financial planning, money management, and investment strategies. However, it was limited by the complexity of the Second Life platform itself. Navigating the virtual environment and interacting with its scripting language could present a steep learning curve, making the game less accessible to some users. Additionally, its approach to failure which required multiple attempts before success to lead to frustration and demotivation will reduce its educational effectiveness. While the game offers strong foundational learning, it might also lack flexibility to incorporate more advanced financial literacy content as players progress.

The ***Wealth Creation***, developed by Kasetsart University, offered a structured and strategic approach to trach financial literacy through resource management, investment decision-making and risk management. The game's strength lied in simulating real-world financial choices within a well-organized and accessible framework. However, it also relied on a simplified financial model. While it introduced key financial concepts effectively, it did not fully capture complex realities such as fluctuating interest rates, inflation or the influence of geopolitical events on financial markets. Although the game emphasizes strategic decision-making, it may not provide a wide enough variety of financial challenges that potentially limit the development of more nuanced financial literacy skills.

Chapter 3

System Methodology/Approach

3.1 System Design Diagram



Finance Journey In University - System Design Diagram
Start Game
Create Charater Customization
Enter to main Scene
Displaying UI for tutorial
Exploration between Scene
Talk to NPC for further infotmation
Player requires to attend to class and sleep due to Time System
Character needs to care status bar to avoid dead
Buy items in Store / play for entertaiment
Buy stock in Stock Market
Earn money via part-time job
Pay bill for survive
Take examination for acedemic performance and feedback at the end of semester
Ending

3-1 *Finance Journey in University* System Design Diagram

The system created for *Finance Journey in University* focused on teaching financial literacy while the main game mechanisms intended to influence the player's experience and the general gaming flow. Before entering the main scenario, players could customise their student avatar during the game's character creation phase. A tutorial interface was then displayed to introduce the fundamental mechanics such as managing time, finances and the character's health-related status bars. Players were encouraged to explore different scenes including the dormitory, university and shop street which served as the central spaces for academic, financial, and social activities.

Players could interact with non-playable characters (NPCs) to gather information and some special events. The time system played a crucial role throughout the gameplay as it was not only dictated the day and night cycle but also influenced the player's responsibilities such as attending lectures, working part-time jobs and ensuring sufficient rest. At the same time, the character's hunger, thirst, health and mental health must be maintained carefully which might led to game over. This created a constant balance between survival needs and academic commitments.

The financial systems provided an additional layer of challenge and realism. Players could earn money through part-time jobs or by taking risks in the stock market. The expenses came in form of bills, food and entertainment to force player to spend the money. The store allowed player to purchase items that directly support survival and well-being whereas failure to pay bills on time will result in penalties or an interrupted end to the game. This system encouraged players to practise financial literacy by managing income, expenses and investment decisions within limited time constraints.

The examination system indicated significant gameplay milestones as the semester progress. Examination was based on attendance requirement that rewarded players with advancement towards a positive conclusion. On the other hand, a poor ending or game-over situation could result from failing tests, ignoring life aspects or not paying payments. The game offered replay value because players could try out various strategies to strike a balance between wealth, health, and academic success repeatedly. In this sense, the system design of game produced a comprehensive simulation of college life where success depended on making wise decisions.

3.1.1 System Architecture Diagram

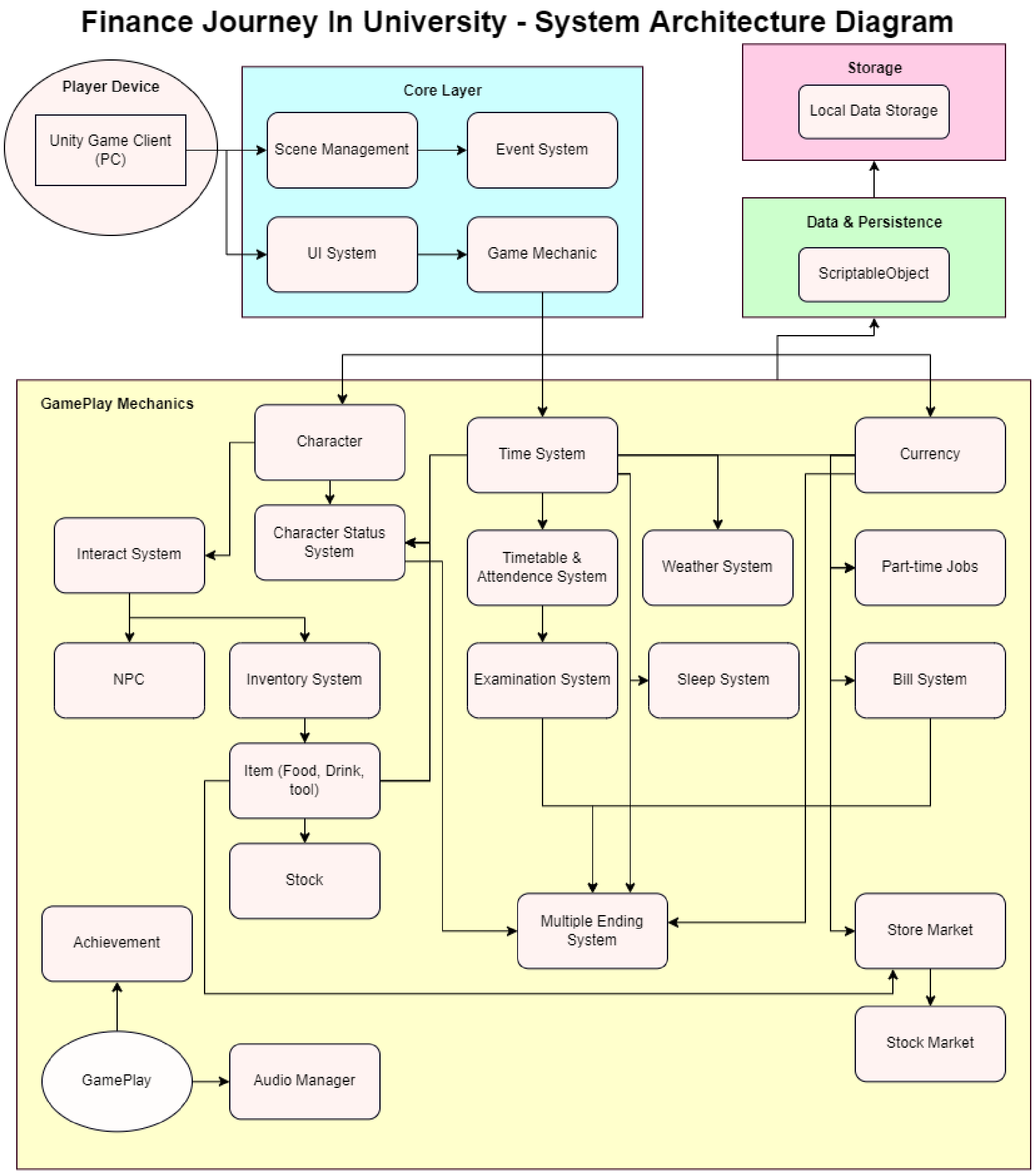


Figure 3-1-1 *Finance Journey in University* System Architecture Diagram

Based on Figure 3-1-1, *Finance Journey in University* structured a layered architecture pattern into five distinct layers. This architecture supported a complex university life simulation game where multiple systems interact to create immersive gameplay experiences centered around financial management and academic success.

The Unity Game Client (PC) served as the runtime environment and target platform. This layer represented the Unity engine instance executing the game on the player's personal computer which handling hardware abstraction and platform-specific operations.

The Core Layer was the application foundation which managing fundamental game operations through four primary components which were Scene Management, Event System, UI System and Game Mechanic. Scene Management handled transitions between game environments including the dormitory, university campus and shopping street in the game. It coordinated with the transition system to provide better navigation experiences. Event System operated as the central communication hub which dispatching events between disparate systems and maintaining loose coupling throughout the architecture. UI System managed user interface elements, handling input processing, screen updates and player interaction feedback across all game screens and overlays. Game Mechanic served as the core mechanism that ensuring gameplay systems were fine working to establish game mechanics and maintaining consistency across all player interactions.

The GamePlay Mechanics Layer contained the business logic which was separated into five interconnected subsystems that drove the core gameplay experiences. The Character Management subsystem centred on the Character entity as the primary player representation and was supported by the Character Status System which maintained four critical statuses such as hunger, thirst, health and mental health. These values were reduced over time to create constant resource management pressure.

The Time Systems subsystem operated through the Time System which handled the day/night cycles and triggered time-dependent events while coordinating with the Timetable and Attendance System for class schedule management. Meanwhile, it had also managed the Weather System for environmental condition generation and the Examination System for critical assessment events on predetermined game days.

The Economic Systems subsystem managed all financial aspects through the Currency System which acted as an important role in game. This subsystem integrated Part-time Jobs for income generation, the Bill System for recurring financial obligations that could trigger failure states and dual market systems including the Store Market for essential purchases and the Stock Market for investment opportunities with dynamic pricing.

The Inventory and Item Management subsystem provided comprehensive item handling through the Inventory System which managed player storage capabilities and the Item System, which organised items into four categories which were food, drinks, tools and stock. Within this, the Stock subsystem specifically managed tradeable items with fluctuating market values.

Finally, the Social and Progression subsystem encompassed player with world interactions through the Interact System which coordinated with NPC entities. The Achievement System tracked progress across multiple metrics while the Sleep System managed rest mechanics that affected character status recovery.

In data and persistence phase, ScriptableObject components stored game configuration data including item definitions, NPC dialogue trees, examination parameters and system balance values. This Unity-native solution provided efficient data management and supported easy content modification without requiring code changes. Local Data Storage handled persistent data such as saved game files, player preferences, achievement progress and system configurations. This layer ensured game state preservation across play sessions and maintained player progression data.

3.1.2 Use Case Diagram and Description

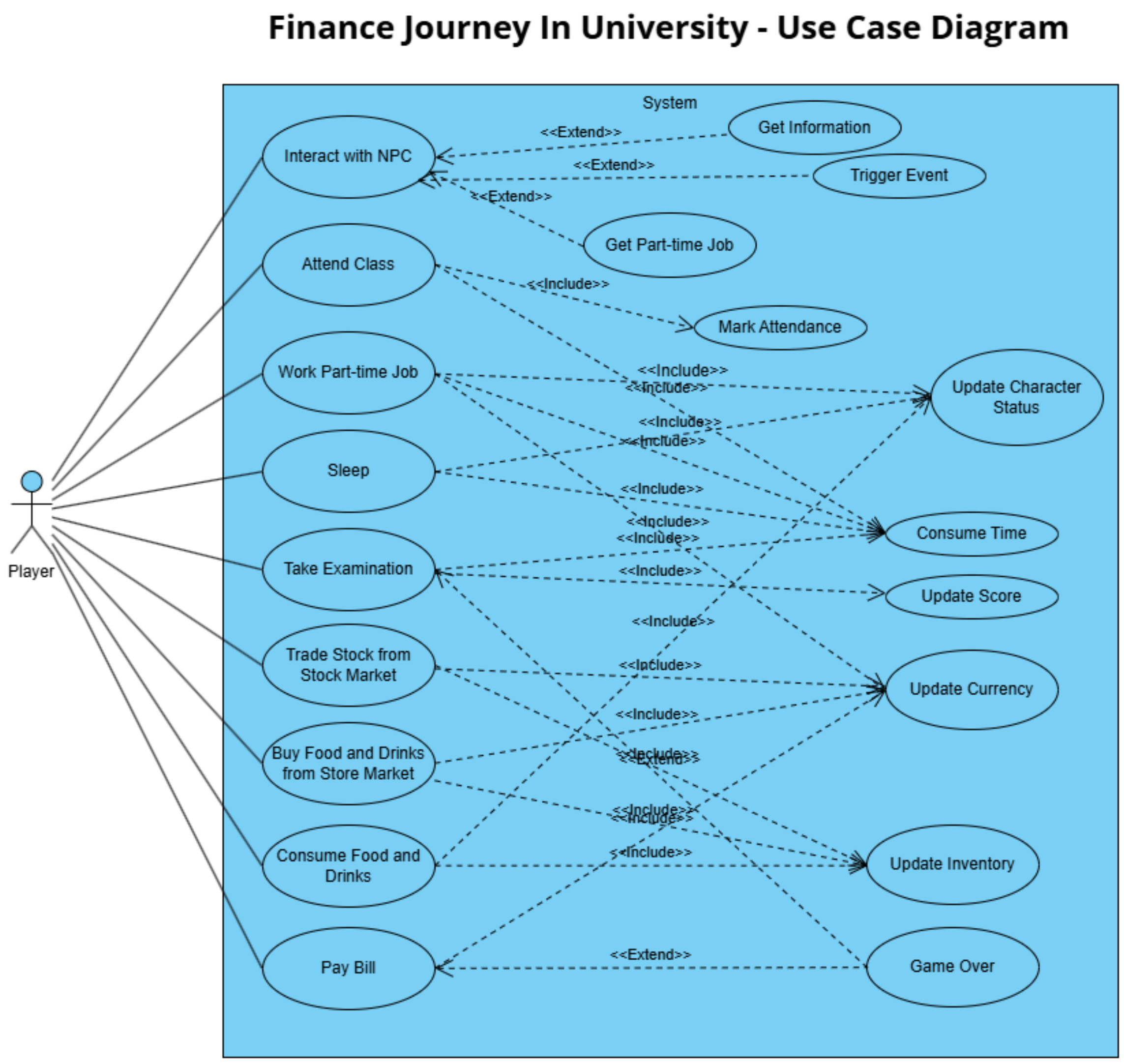


Figure 3-1-2 *Finance Journey in University* Use Case Diagram

In the use case diagram for this simulation game, the central actor was the Player who interacted with various features of the system to progress through university life. Each use case represented a significant gameplay activity while the use of include and extend relationships showed the dependencies and optional outcomes.

The Interact with NPC use case was extended by three optional scenarios which were trigger event, get information and get part-time job. This reflected the fact that conversations with non-playable characters could lead to different results, some of which were not guaranteed in every interaction. Similarly, the attend class use case included both mark attendance and consume time because these were necessary outcomes of attending a lesson.

The work part-time job use case included consume time, update character status such as decreasing mental health and updating currency. These actions were carried out whenever the player chose to work, as jobs both rewarded money and consumed time. Moreover, the sleep use case included update character status along with covering changes in health, mental health, hunger and thirst as well as consuming time which highlighting its restorative but time-consuming nature.

The take examination use case was modelled with both include and extend. It included updating score and consuming time since they were essential to completing an exam while it extended to game over as failing the final examination could terminate the game and lead to one of the possible endings.

In terms of resource management, trade stock from the stock market included update inventory and update currency, since buying or selling assets always affected these. In addition, buying food and drinks from the store market also included updating both inventory and currency, reflecting the real experience of shopping.

Finally, the pay bill use case included update currency because money was always deducted, while it extended to game over if the player failed to pay bills on time. This emphasised the critical role of financial management in the gameplay loop.

3.1.3 Activity Diagram

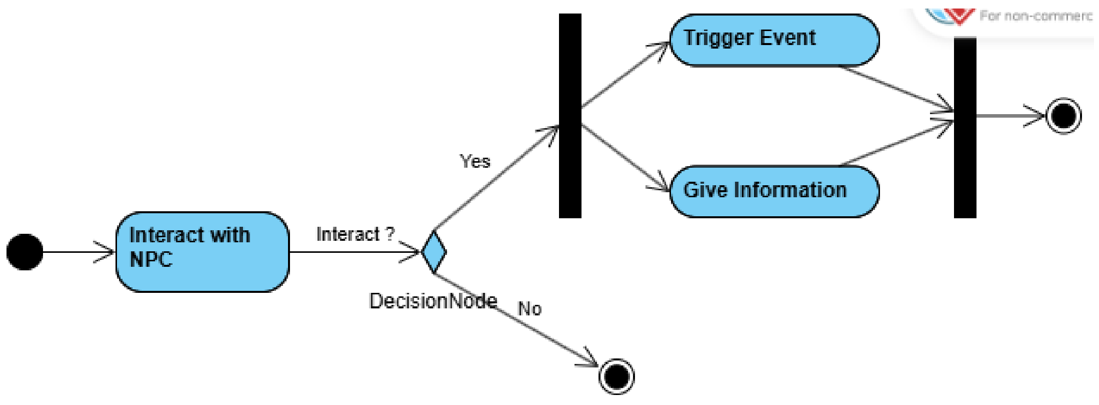


Figure 3-1-3 Interact with NPC Activity Diagram

Based on Figure 3-1-3, activity diagram interacted with NPC use case shows the sequences when player intended to interact with NPC. Once the interaction was confirmed, both activities were triggering event and providing information that performed concurrently to update the situation in game. The action will be cancelled if player rejected the interaction.

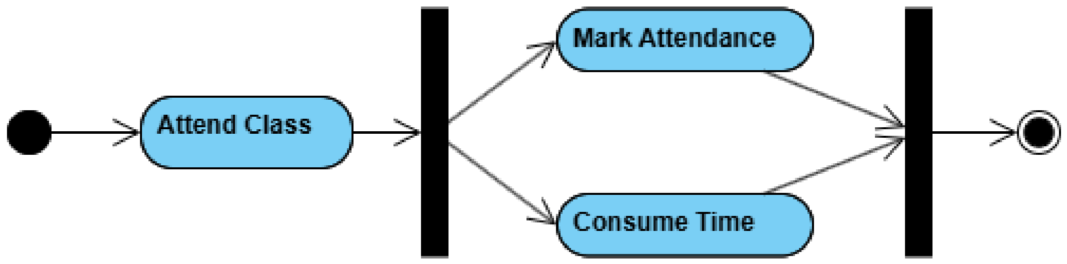


Figure 3-1-4 Attend Class Activity Diagram

When a player attended class in the game, the attendance was marked and time was consumed in the game. The attendance was recorded in the attendance system and update in terms of percentage which could be viewed by player through UI interface.

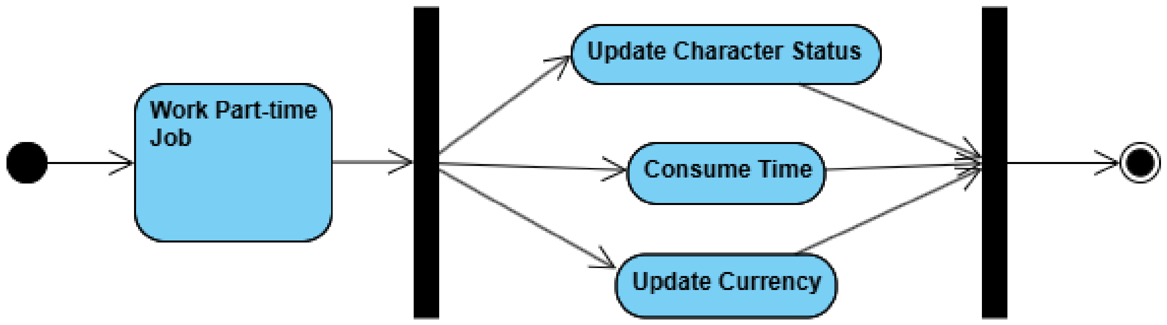


Figure 3-1-5 Work Part-time Job Activity Diagram

A player could increase the amount of money by working at part-time jobs in the game application. The character's status like mental health decreased due to exhaustion from working and time passed in a fixed unit measured in hours.

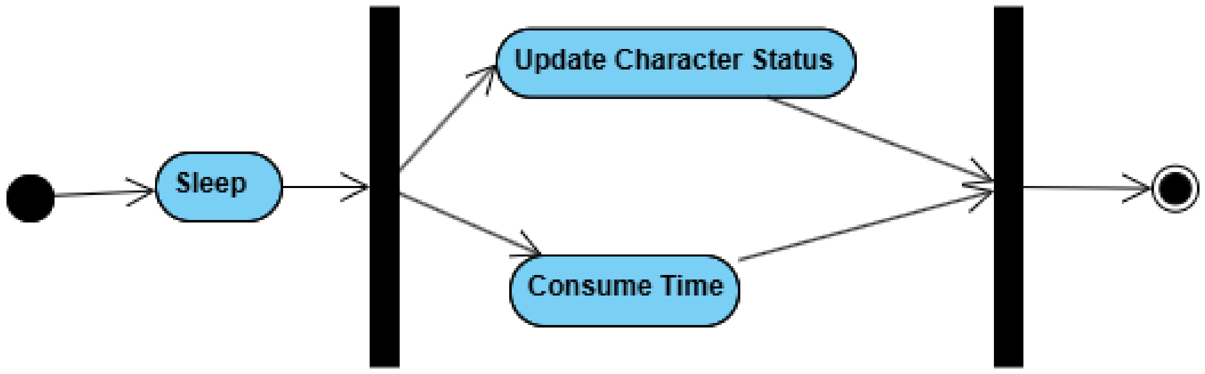


Figure 3-1-6 Sleep Activity Diagram

The activity diagram for the sleep use case presented the process that occurred when the player chose to make the character sleep. Once the action was initiated, the workflow split into two parallel activities. The first activity updated the character's status such as restoring energy or health while the second activity consumes in-game time by moving the game clock forward. Both activities occurred simultaneously and were synchronized before the process ended. This ensured that the character's condition improved and in-game timeline progressed consistently whenever the sleep action was performed.

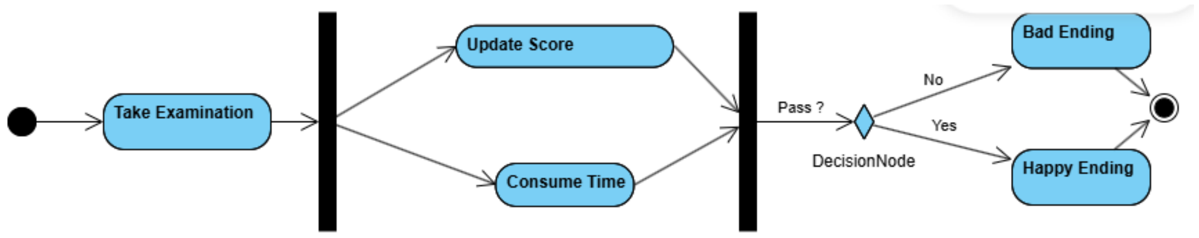


Figure 3-1-7 Take Examination Activity Diagram

The activity flow for the take examination process began with the student taking the examination. Once the exam was initiated, two concurrent activities occurred. The score was updated and time was consumed which reflected both academic result and time management aspects of the process. After these activities were completed, the flow proceeded to a decision node that evaluated whether the student had passed or failed the examination. If the outcome was with a good academic result, the process led to a "Happy Ending" representing success and a positive result. On the other hand, if the student failed then the flow concluded with a "Bad Ending" representing an unsuccessful outcome.

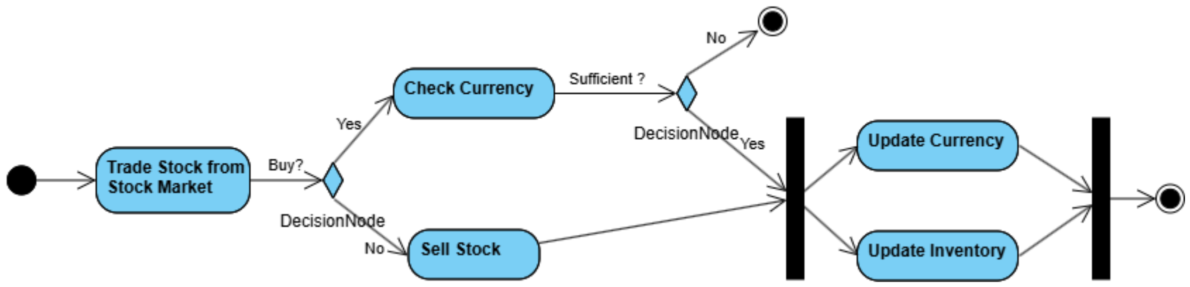


Figure 3-1-8 Trade Stock Activity Diagram

By following Figure 3-1-8 visualised the process of use case of trading stock from stock market, checking currency action was performed for further execution while player was intended to buy stock. Two activities were executed concurrently when the money was sufficient to buy the stock which updated the amount of currency and added the stock into player's inventory. On the other hand, player was allowed to sell stock from inventory to earn money and the stock will be deducted from the inventory. In the end, the process was once buying or selling stock was completed.

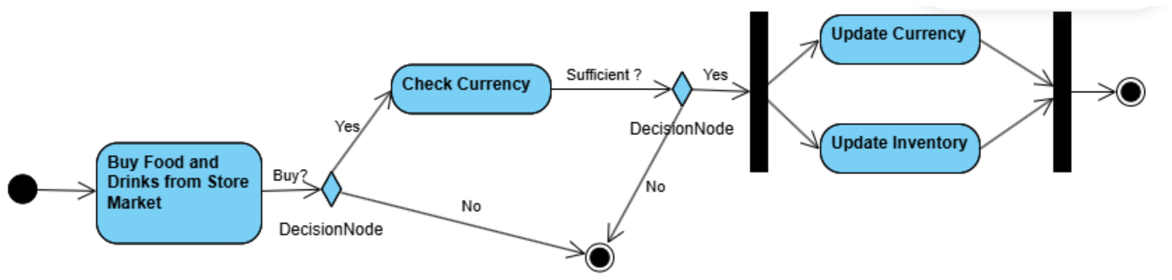


Figure 3-1-9 Buy Food and Drinks Activity Diagram

The activity diagram from Figure 3-1-9 illustrated the process of purchasing food and drinks from the store market. The sequence commenced when user initiated the purchase. A decision node determined whether the user chose to buy or not. If the decision was negative, the process concluded immediately. If the decision was positive, the system proceeded to check the available currency. In cases where the currency was insufficient, the process terminated. However, when the currency was adequate, the system carried out two concurrent activities: updating the currency to reflect the deduction and updating the inventory to include the purchased items. Once these activities were completed, the process successfully ended.

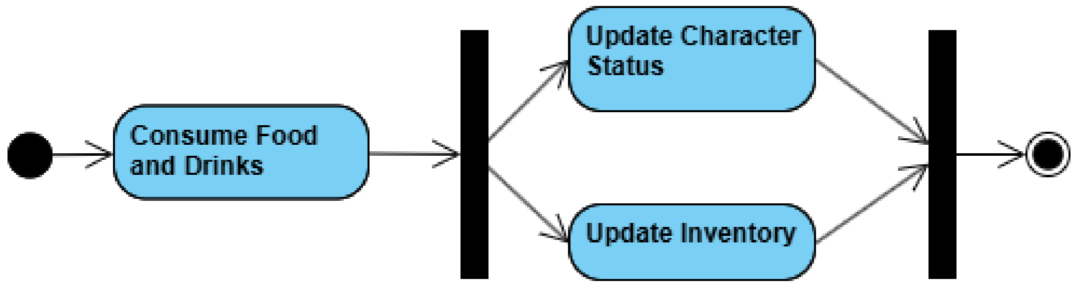


Figure 3-1-10 Consume Food and Drinks Activity Diagram

Based on Figure 3-1-10, the activity diagram discussed the process of consuming food and drinks within the system. The sequence began when the character consumed the selected items. Following this action, two concurrent activities were performed which the system updated the character's status to reflect the effects of the consumption and the inventory was adjusted to remove the consumed items. Once both updates were completed, the process successfully completed the mission.

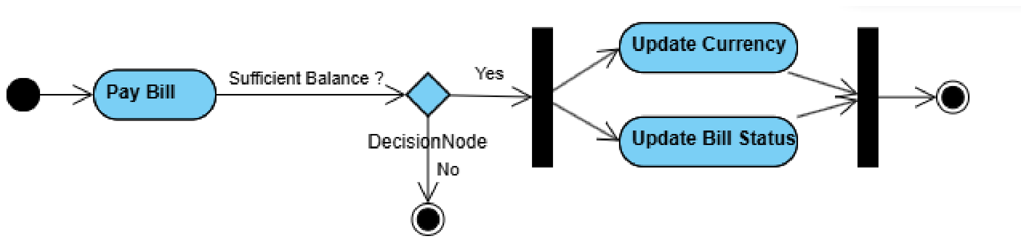
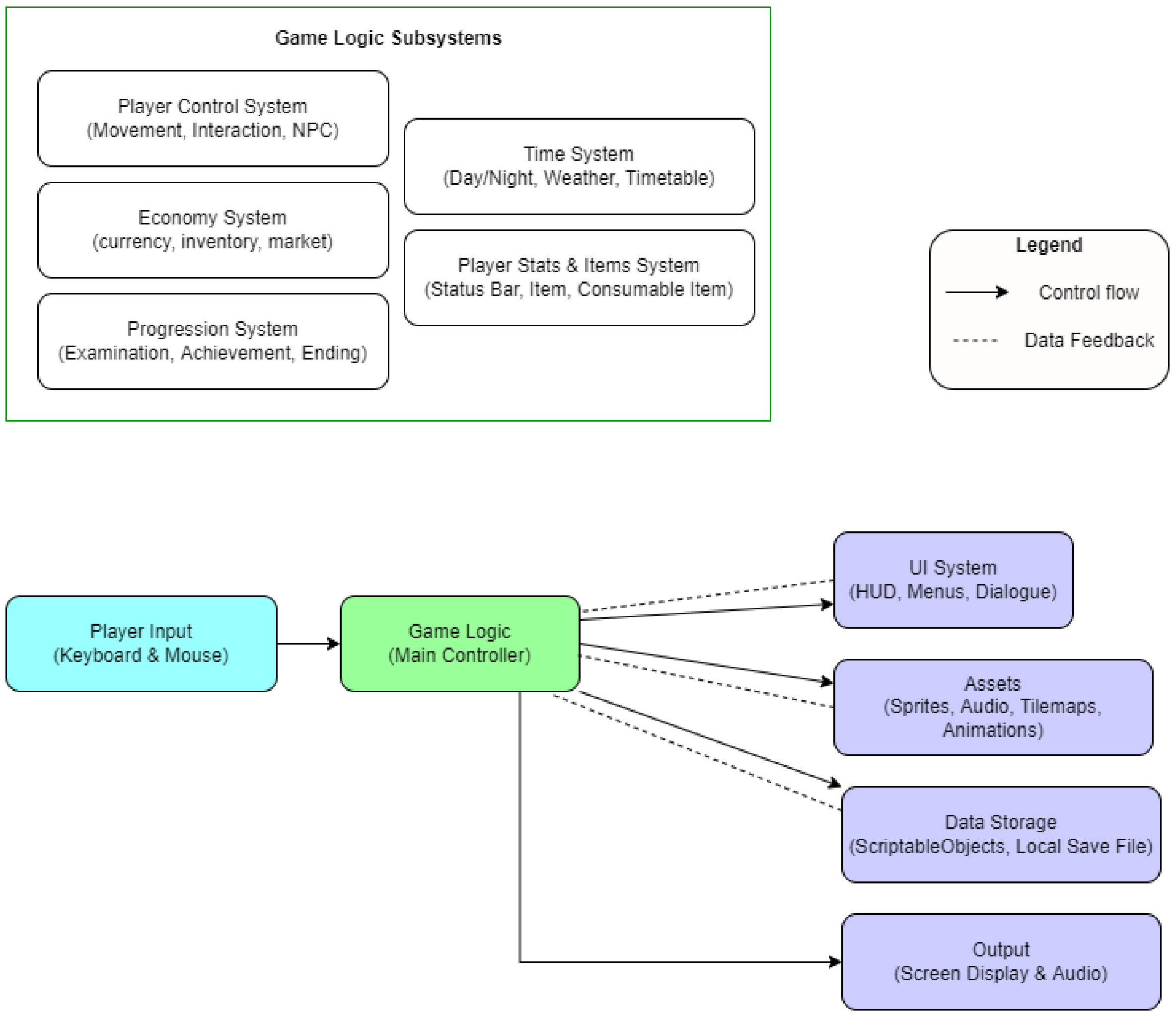


Figure 3-1-11 Pay Bill Activity Diagram

Activity diagram above showed the process of paying a bill. It began with the user attempting to pay, followed by a decision node that checked whether the balance was sufficient. If the balance was not enough, the process ended immediately. However, if the balance was sufficient, the flow continued to two concurrent activities: updating the currency and updating the bill status. Once both activities were completed, the process ended successfully. This diagram illustrated how the system ensured payments were only processed with enough balance and updated records accordingly.

Chapter 4
System Design

4.1 System Block Diagram



Game Logic Subsystems
Player Control System (Movement, Interaction, NPC)
Economy System (currency, inventory, market)
Progression System (Examination, Achievement, Ending)
Time System (Day/Night, Weather, Timetable)
Player Stats & Items System (Status Bar, Item, Consumable Item)
Legend
Control flow
Data Feedback
Player Input (Keyboard & Mouse)
Game Logic (Main Controller)
UI System (HUD, Menus, Dialogue)
Assets (Sprites, Audio, Tilemaps, Animations)
Data Storage (ScriptableObjects, Local Save File)
Output (Screen Display & Audio)

Figure 4-1-1 Finance Journey in University System Block Diagram

Figure 4-1-1 presented the system block diagram of the developed 2D top-down educational game Finance Journey in University. The system was designed by top-down approach where the main controller was represented by game logic block. All game mechanics were processed through this central controller which coordinated with subsystems, user input, assets, storage and output.

The first block system began with player input block where keyboard and mouse were the essential devices for input to control character movement, interaction and decision making. The input value was passed to game logic block once player input was updated by player. Game logic block that consisted of multiple integrated subsystems like player control, economy, progression, time and player stats & items system will process the input value for specific functionality. Each system handled specific important responsibility in the gameplay. For example, the economy subsystem handled currency which was money, inventory, storage and market transactions while time subsystem managed the day and night cycle together with special event triggers. Player control system was a system that related to player input block, it handled the logic of interaction between player character and game object in the whole gameplay like character movement, item and NPC interaction. Player stats & items system used to enhance the game experience and realism towards player and increase the difficulty of the game application. By tracking player's progress in the game, the progression subsystem was implemented to respond to player's gameplay process and reflect the actual knowledge and skill that player applied in the entire game. It also helped to examine how player knew about financial literacy knowledge and told that how much effort player spent to gain the knowledge.

Moreover, the game logic interacted with other backend and frontend core components like UI system. The UI System functions as the main bridge between the Game Logic and the player by providing real-time visual and interactive feedback. It presents core gameplay information through heads-up displays, menus, dialogue panels, and popups. For example, the UI system displays the player's status bar, time panel, and store panel while also providing multiple options for decision-making and daily activities.

The Assets block contains all the visual and audio resources required for the game, such as sprites, tilemaps, animations, sound effects, and background music. These assets work in coordination with the Game Logic and UI System to create an immersive 2D top-down experience. Assets were managed and optimized within Unity to ensure smooth performance and consistent visual effect. Sprites and tilemaps acted as very important elements in the game which built the scene, character model and item appearance. Sprites were used to build numerous of character models, items and UI interface which attracting player's attention while tilemaps were used to build the whole scene design called "map" with multiple layers to show the depth of building or furniture in the scene in natural way.

The Data Storage block was responsible for maintaining and retrieving essential information that supports persistent gameplay. This includes inventory, financial records and in-game events. ScriptableObjects and file-based storage in Unity were used to manage structured game data, ensuring both flexibility and efficiency in data handling.

Finally, the Output block represents the visual and audio delivery of the game through the player's monitor and speakers. It translates processed game logic, assets, and UI components into an interactive audiovisual experience. The audio like background music used for expressing the whole tone in the game and sound effects were used for feedback from interaction and gave information to player. The output ensures that every action, decision, and progression in the game was clearly communicated to the player.

4.2 System Components Specifications

The developed game utilised the Unity engine as the main development platform and was constructed with multiple integrated components. The environments of the game were built primarily with Unity's Tilemap system.

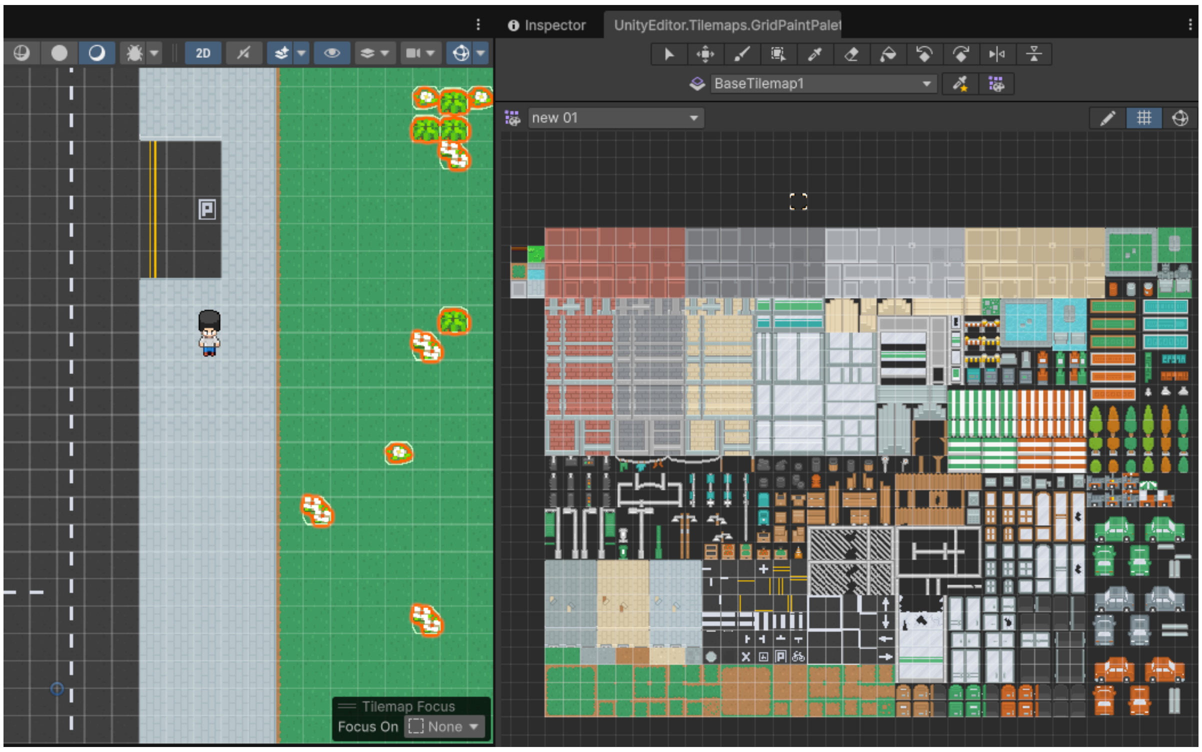


Figure 4-2-1 Grid Paint Palette in Unity

Figure 4-2-1 illustrated the Grid Paint Palette and Rule Tile functions provided within the Unity editor allowed the design of scenes to be carried out efficiently. Rule Tiles were particularly useful for automatically assigning the correct edges and corners of tiles, which reduced the time required to design maps. Several scenes were designed during development, including the exterior of the university building known as Block N, the interior corridors of Block N, various classrooms, a shop street, and the dormitory.

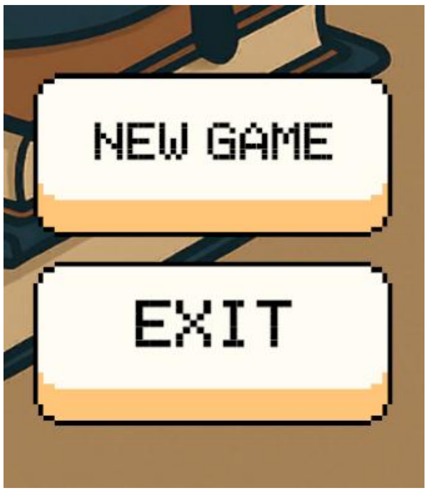


Figure 4-2-2 Option Buttons for Main Menu Scene

The game began with a main menu scene that contained only user interface elements like "New Game" button and "Exit" button. This scene allowed player to start the game by clicking on the buttons and create character based on their preference.

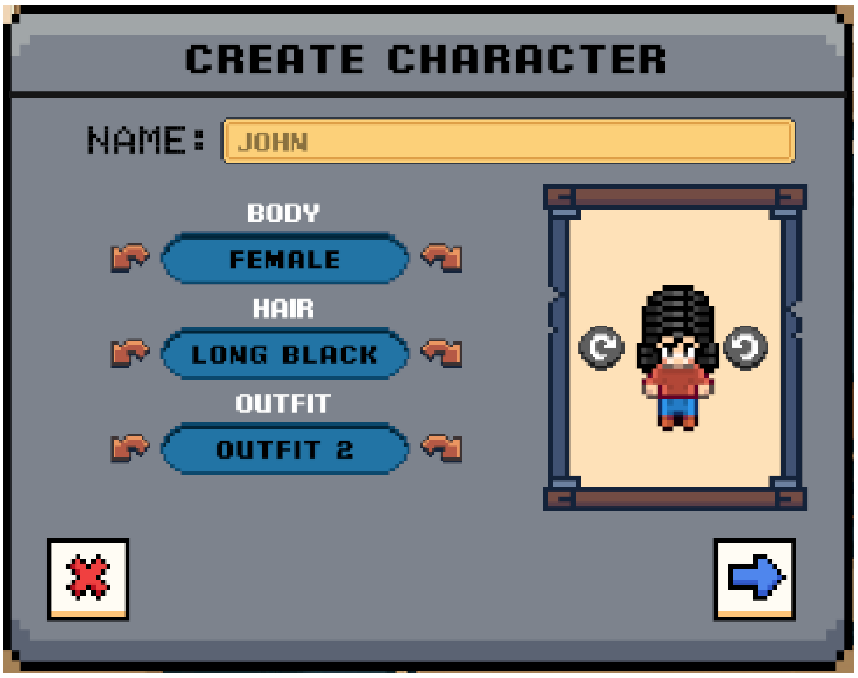


Figure 4-2-3 Character Creation Customization

Figure above showed the process of character creation, it allows player to decide the player data which were name and appearance of character. The body of character could be set as male or female as a gender identity, long or short hair options were given to player and two outfits with different appearance.

In addition, scene management was handled using Unity's Scene Manager, which enabled transitions between scenes. The system was configured so that the game always loaded two scenes at once: a place scene and an essential scene. The place scene was the environment in which the player explored, such as a classroom or shop. The essential scene was persistent across all transitions and contained the player data, the user interface canvas, the game manager and other systems required to be maintained throughout the gameplay. The system for scene transitions was achieved using a transition script attached to colliders within the environment. When the player entered the collider, the script was triggered, which loaded the new place scene additively while retaining the essential scene in memory.

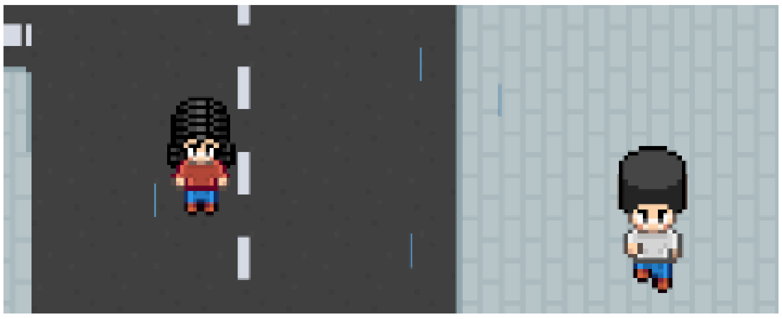


Figure 4-2-4 Character Standing in the Game

The player character itself was implemented using a Rigidbody2D component with rotation frozen on the Z axis to avoid unwanted spinning. A custom character controller script determined the walking speed and running speed of the character. The movement of the player was handled through the WASD keys, while holding the shift key increased movement speed to simulate running. The animator controller was used to blend animations smoothly, two parameters were passed from the character controller to the animator which were the movement state and the last motion vector. These ensured that the correct animation was displayed when the player moved and that the character continued to face the correct direction when standing still. Collision detection was implemented with a 2D box collider, which prevented the character from passing through walls and furniture and enabled interactions with non-playable characters (NPCs) and objects. When interacting with chests or items, the system opened the relevant user interface panel and processed the player's action. Entering new areas or opening shop panels was managed by colliders configured as triggers. The transition script responded to these triggers and brought the player to the new location.

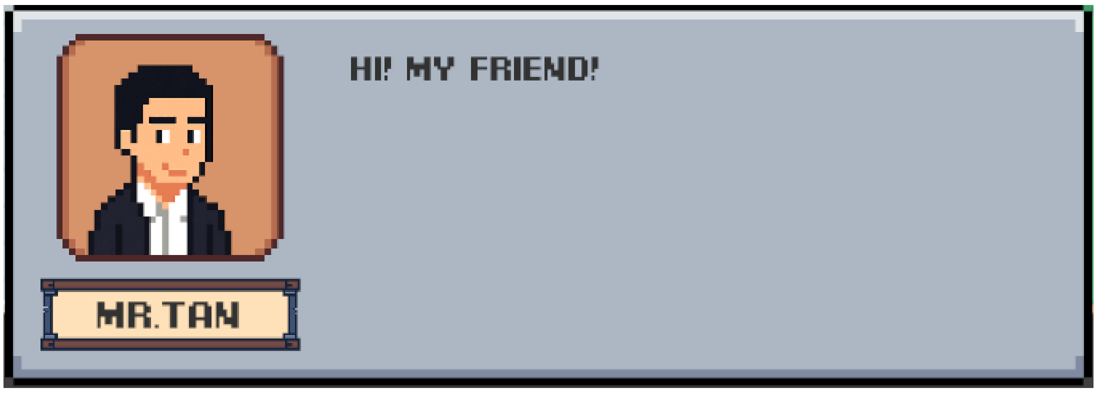


Figure 4-2-5 Dialogue Panel

Interactions with objects and NPCs were triggered by the player through right mouse clicks. For example, a dialogue panel will show up when player right-click to talk to NPC in game which shown by Figure 4-2-5.

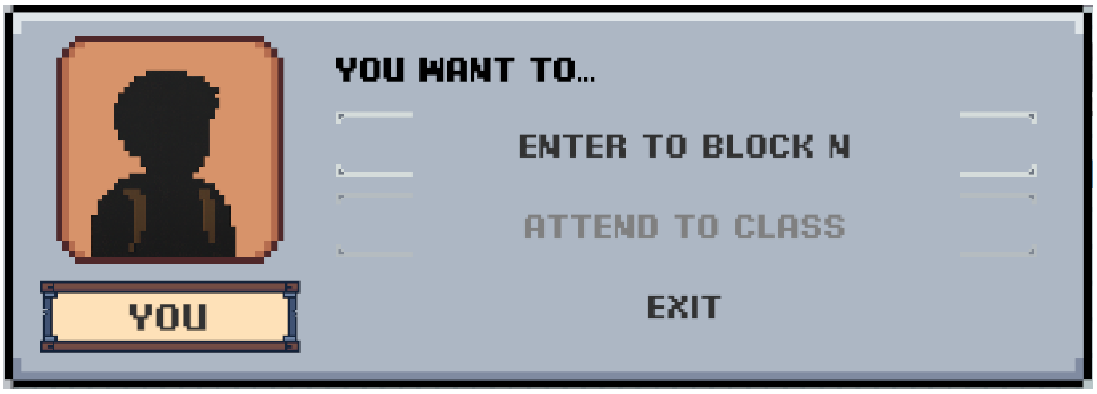
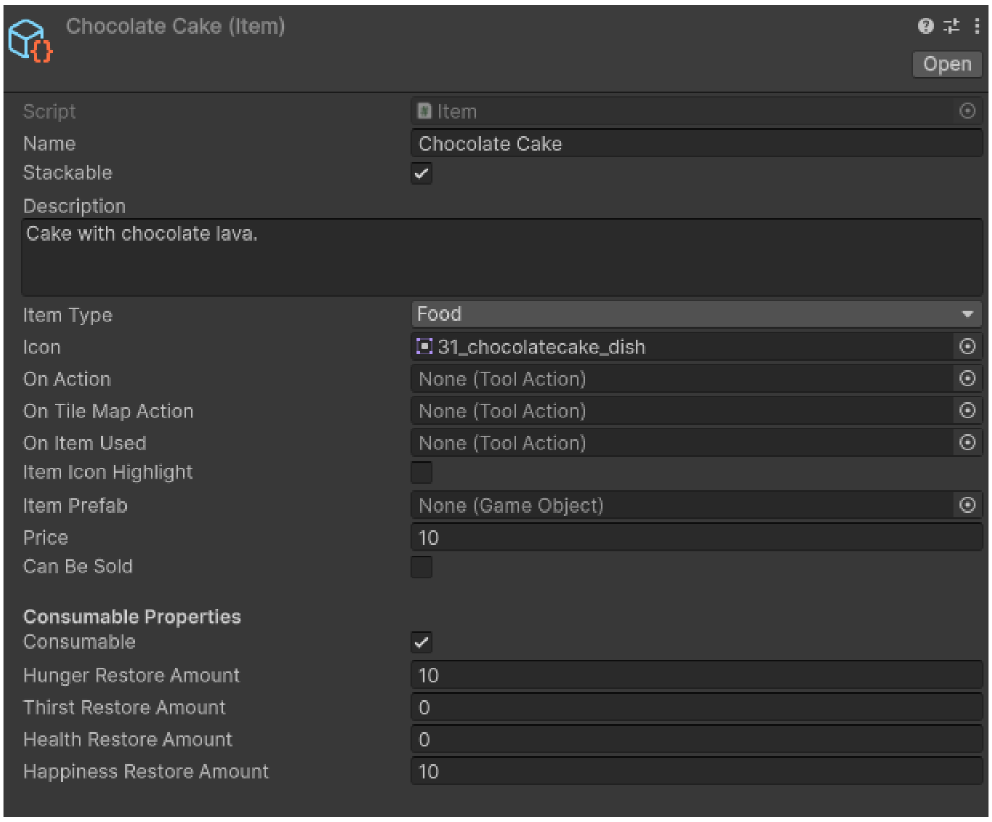


Figure 4-2-6 Option Dialogue Panel

Option dialogue panel was advanced version of dialogue panel which allowed player to make decisions to achieve multiple functionalities by clicking on the option. Each option was able to configure independently by specific requirements or conditions.

A data-driven approach was adopted for the management of in-game entities through Unity's ScriptableObject system. Different ScriptableObject definitions were created, including items, stock, dialogues, option dialogues, inventories, item containers, store inventories, stock store inventories, non-playable characters (NPC) and player character body parts.



Chocolate Cake (Item)

Open

Script	Item
Name	Chocolate Cake
Stackable	✓
Description	Cake with chocolate lava.
Item Type	Food
Icon	31_chocolatecake_dish
On Action	None (Tool Action)
On Tile Map Action	None (Tool Action)
On Item Used	None (Tool Action)
Item Icon Highlight	
Item Prefab	None (Game Object)
Price	10
Can Be Sold	
Consumable Properties	
Consumable	✓
Hunger Restore Amount	10
Thirst Restore Amount	0
Health Restore Amount	0
Happiness Restore Amount	10

Figure 4-2-7 ScriptableObject Configuration in Unity Inspector

These ScriptableObjects stored essential data such as names, prices, descriptions and other configurable settings. They were particularly important in ensuring that progress and data were persistent and could be reused across scenes. The example in Figure 4-2-7 showed the item ScriptableObject with some settings like name, stackable, description, icon and some basic settings. The consumable properties in figure should the status change of character's status bar if the item was consumed.

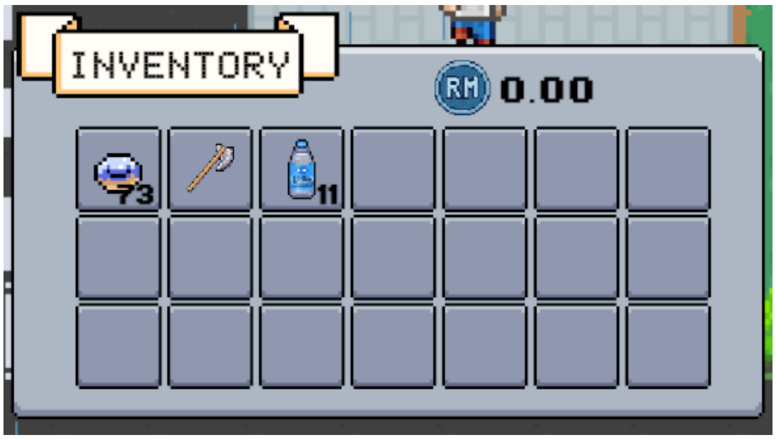


Figure 4-2-8 Inventory System

The figure 4-2-8 showed that inventory system was slot-based and supported drag-and-drop operations. The player could move an item by left clicking and dragging it to another slot or dropping it onto the floor. Right clicking on a stack of items split it into two smaller stacks. The currency system was money system that was viewable by opening the inventory user interface. It stored the current amount of money held by the player and was used in both normal store transactions and stock trading. Additional income could be obtained by taking part-time jobs.

The system also incorporated two specialised financial components which called Store system and the Stock Market system. Both were designed to enhance the financial learning objectives of the game while maintaining consistency with the overall interactive design.

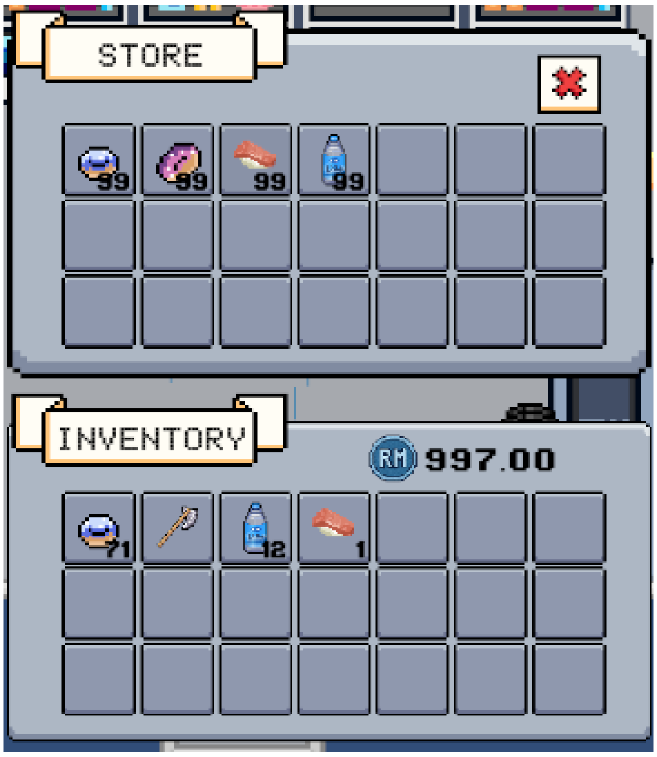
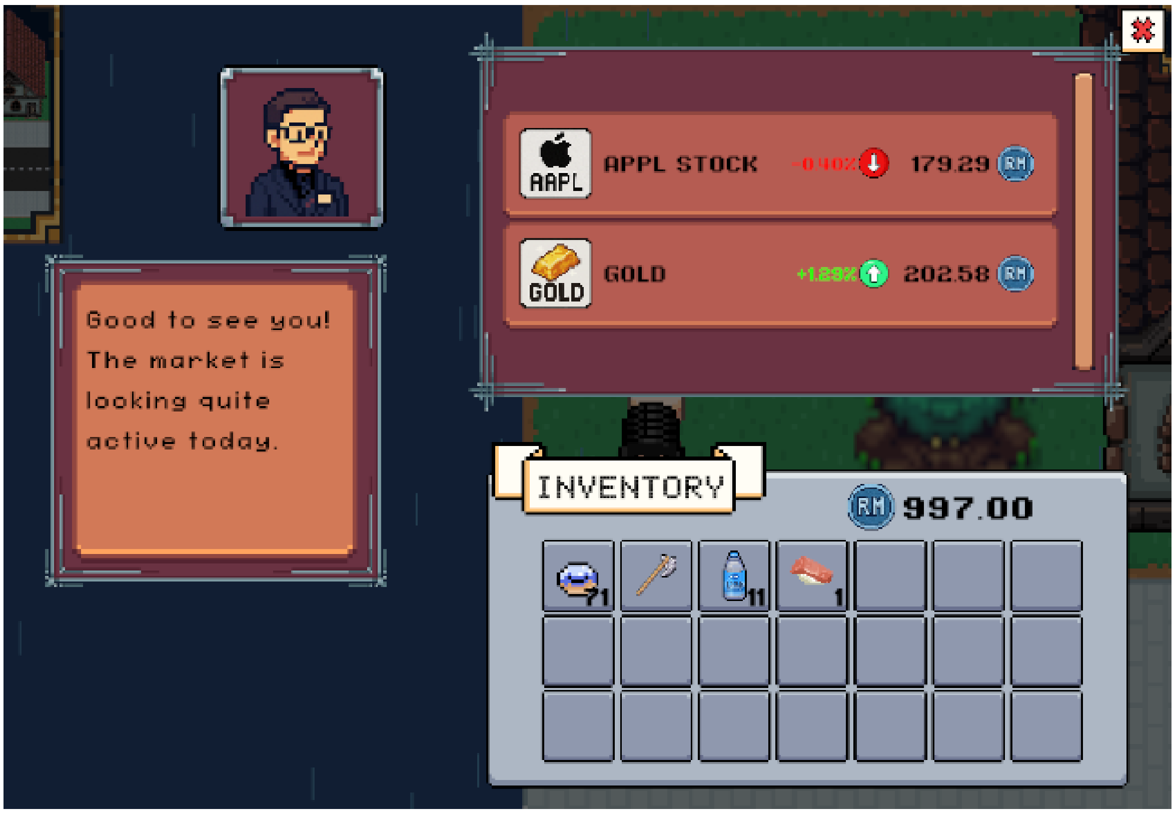


Figure 4-2-9 Store Panel with Inventory Panel

The store system allowed players to purchase consumable and utility items within the game world. Players were able to interact with a designated store clerk non-playable character (NPC) to open the store panel. Figure 4-2-9 presented store panel was implemented as a user interface element that displayed available and always be with inventory system panel. Transactions were executed through integration with the inventory system and currency system, ensuring that purchased items were automatically added to the player's inventory and the corresponding balance was deducted from their wallet. To simulate realism, the store operated on a time-based constraint where the store was opened between 8.00 a.m. and 10.00 p.m. only.


Good to see you! The market is looking quite active today.

AAPL	APPL STOCK	-0.40%	179.29 RM
GOLD	GOLD	+1.29%	202.58 RM

Figure 4-2-10 Stock Store Panel with Inventory Panel

The stock market system was implemented as a more advanced financial interaction mechanism. Players could access this system by engaging with the Bank Agent NPC. The stock store panel was presented through a separate user interface distinct from the store panel. The panel showed a portrait of the Bank Agent alongside with simple dialogue to enhance immersion and contextual learning. The stock interface displayed a list of available stock assets with each accompanied by an arrow icon that indicated whether the value had increased, decreased, or remained unchanged compared to the previous price. The fluctuation of asset values was controlled by a MonoBehaviour script which recalculated percentage-based price changes at fixed intervals. The stock store was available only between 9.00 a.m. and 5.00 p.m. in the in-game time cycle, to reflect the real-world behaviour of financial markets.

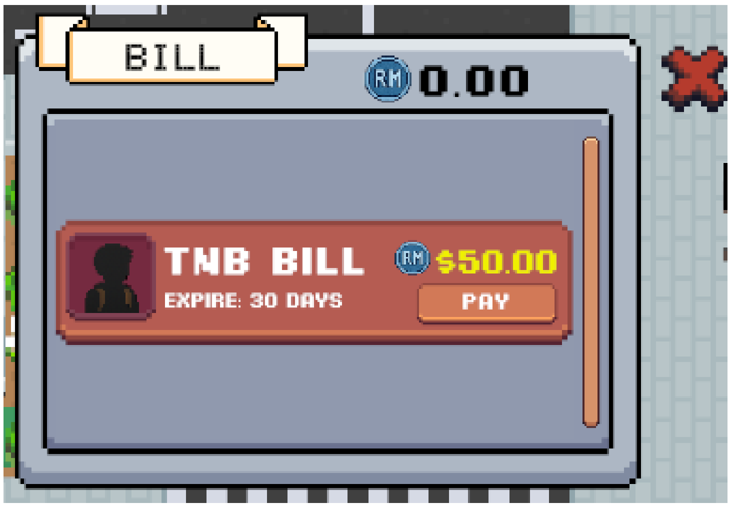


Figure 4-2-11 Bill System

A bill system was also implemented which required the player to pay outstanding amounts before specified deadlines. If the deadline passed without payment, the game concluded with a failure state.

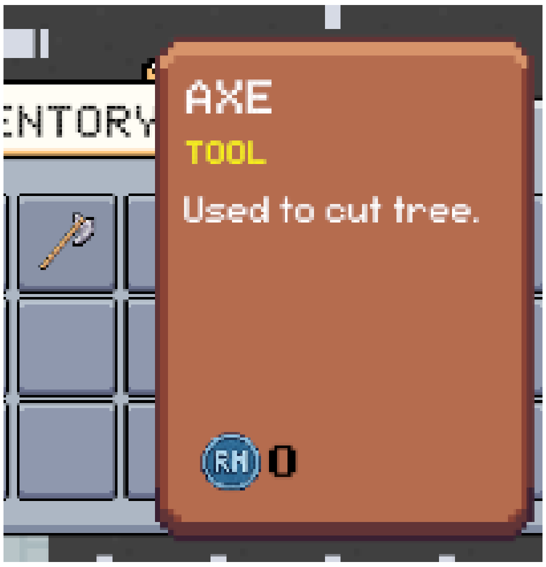


Figure 4-2-12 Item Detail Panel

The object of the game was divided into two categories of items. The regular items, such as food and drinks which consumable or tool that allowed user to perform specific action, were fixed-price goods which could only be purchased but not sold. On the other hand, the stock items were implemented with a fluctuating value that changed every five seconds in game to simulate real-world stock behaviour. This allowed players to engage in buying and selling strategies based on market variation. The action of hovering the mouse over an item triggered a tooltip panel which displayed the item's name, description, and price, as well as any status effects if the item was consumable just like how Figure 4-2-12 showed. The item type will be show in yellow colour to clarify to player to know about the functionality of the item. For example, items category in food or drinks were consumable so that player could right-click to interact and consume with it.

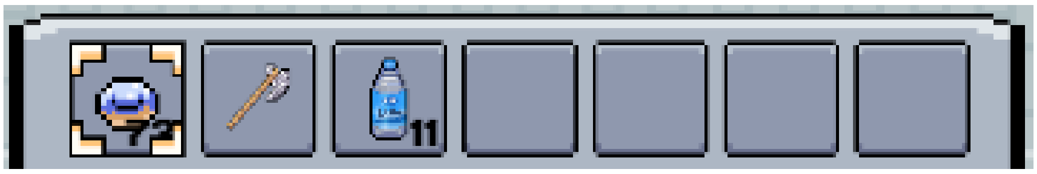


Figure 4-2-13 Tool Bar System

Tool bar system was implemented to let player to select and interact with items just like how the player was taking the items. The interaction was initiated by right-clicking on the mouse device and the scroll on mouse could change the selected item in tool bar system or pressing number key 1 to 7. It was worked closely with inventory system as they were interactable and sharing same item container.

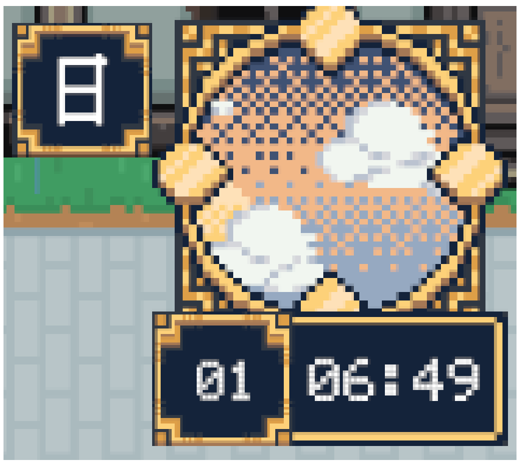


Figure 4-2-14 Time System

Time system played a central role in driving the progress of the game. It presented the day of the week, day passed in the semester and current time in the game based on Figure 4-2-14. Then, Time system simulated a day and night cycle and push the progress of the game to the end when the day passed of the semester was 90. The time system influenced several other subsystems, including the timetable, the bill deadlines, weather conditions, and examination events.

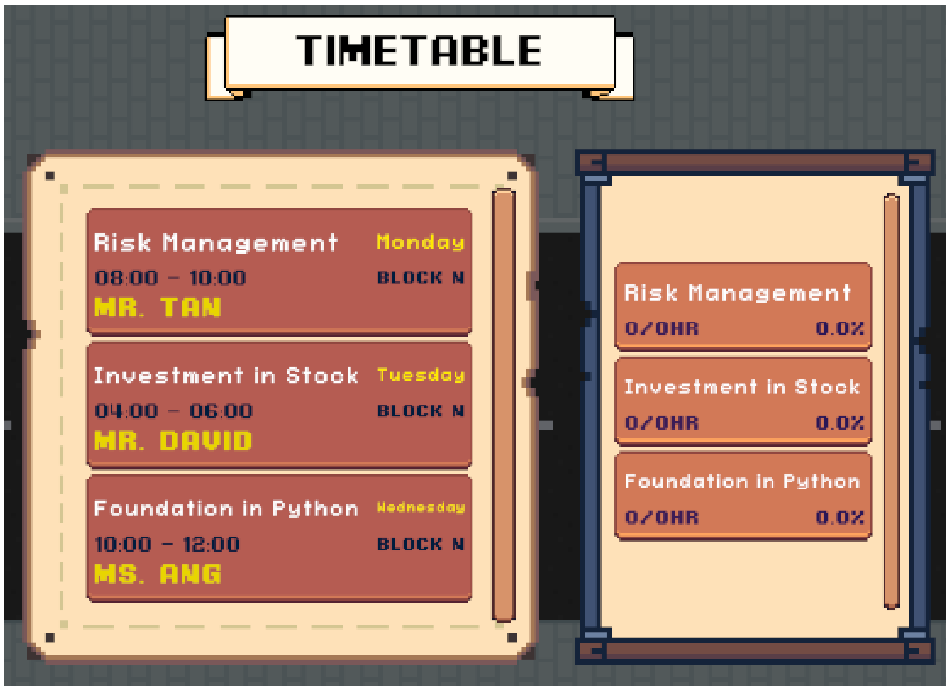


Figure 4-2-15 Timetable System

The timetable system defined the schedule for attending lectures and classes. Attendance was monitored throughout the semester, and failure to maintain at least eighty percent attendance disqualified the player from sitting the final examination. The system also determined when the player must attend the midterm and final examinations, set at the days 45 and days 90 respectively. The player could use the sleep function to skip forward to the next day. The weather system was also linked to the passage of time, allowing randomised sunny, rainy or thunderstorm conditions to be displayed.

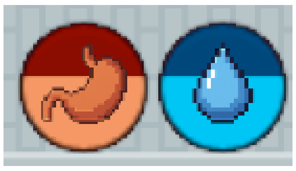


Figure 4-2-16 Hunger and Thirst Status UI

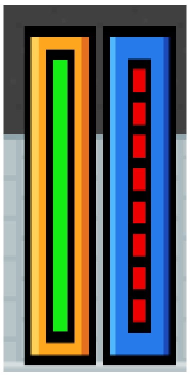


Figure 4-2-17 Mental Health and Health Status Bar

The player character was further supported with a status system consisting of four attributes: hunger, thirst, health and mental health. Hunger and thirst decreased steadily over time, encouraging the player to consume food and drinks. Health was reduced when hunger or thirst reached zero, while mental health decreased if the player worked excessively or lacked sleep. If health or mental health reached zero, the game ended. Consuming food and drinks restored hunger, thirst and mental health, depending on the item consumed.

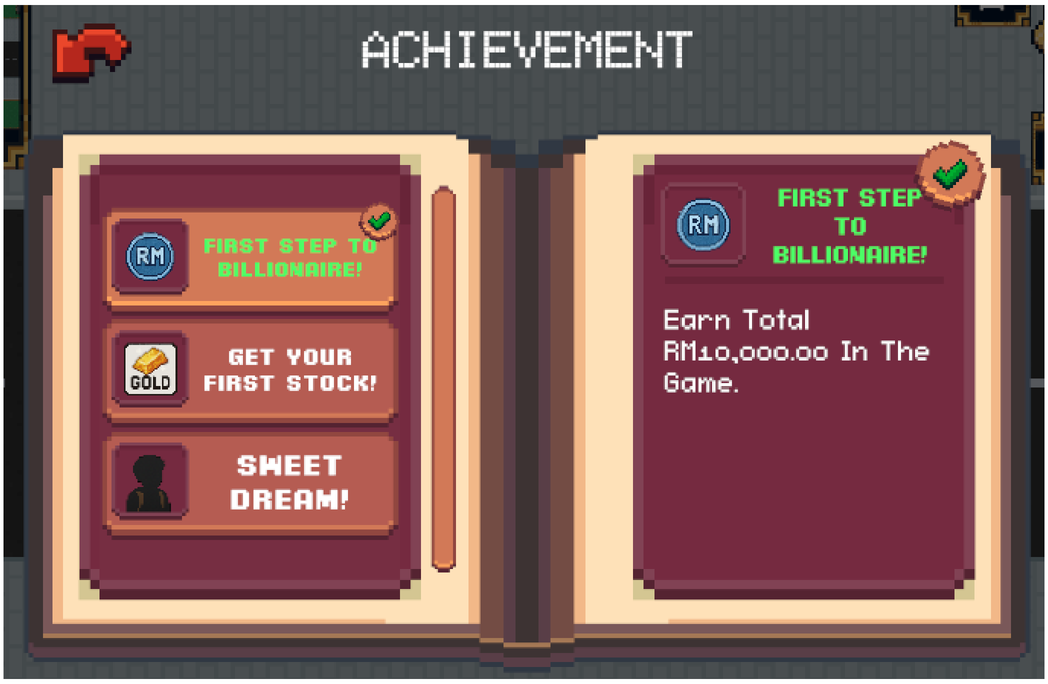

ACHIEVEMENT
FIRST STEP TO BILLIONAIRE!
GET YOUR FIRST STOCK!
SWEET DREAM!
FIRST STEP TO BILLIONAIRE!
Earn Total RM10,000.00 In The Game.

Figure 4-2-18 Achievement System

An achievement system was developed to provide long-term goals for the player. Achievements were predefined in the Unity inspector and stored with unique identifiers. The related script with achievement will check whether achievement conditions had been fulfilled. The left panel of Figure 4-2-18 showed the Achievement with achievement's title and the right panel show the detail of achievement which was description. The complete badge will show in the right top corner of achievement when it was completed.

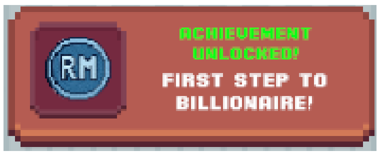
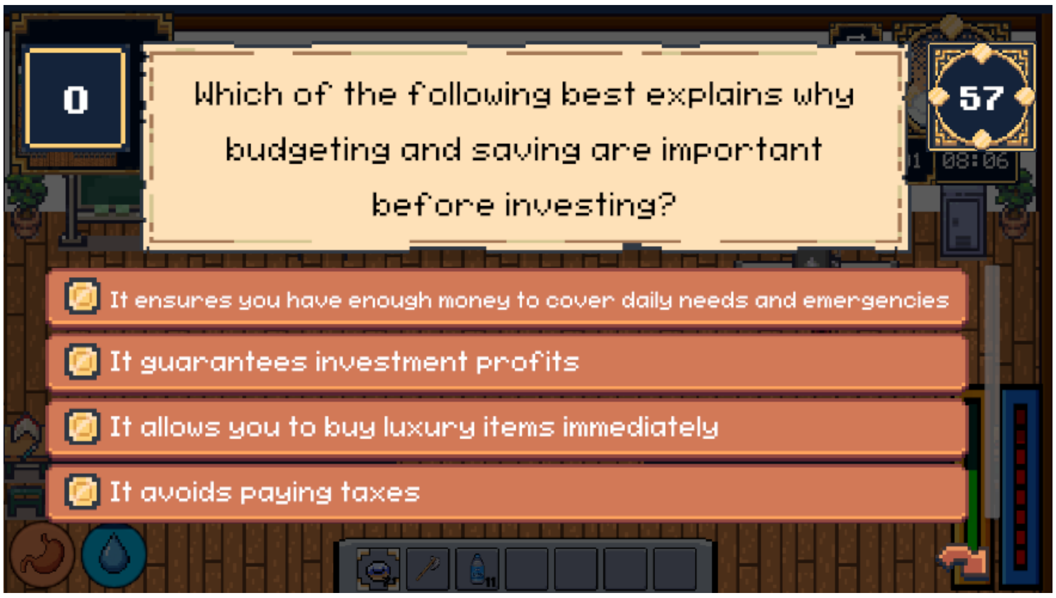


Figure 4-2-19 Achievement Completed Panel

When an achievement was completed, the system saved the progress in PlayerPrefs in Unity and displayed a popup animation on the screen. Achievements could be reviewed through the pause menu, which also provided access to other system options such as resuming or exiting the game.



Which of the following best explains why budgeting and saving are important before investing?

It ensures you have enough money to cover daily needs and emergencies

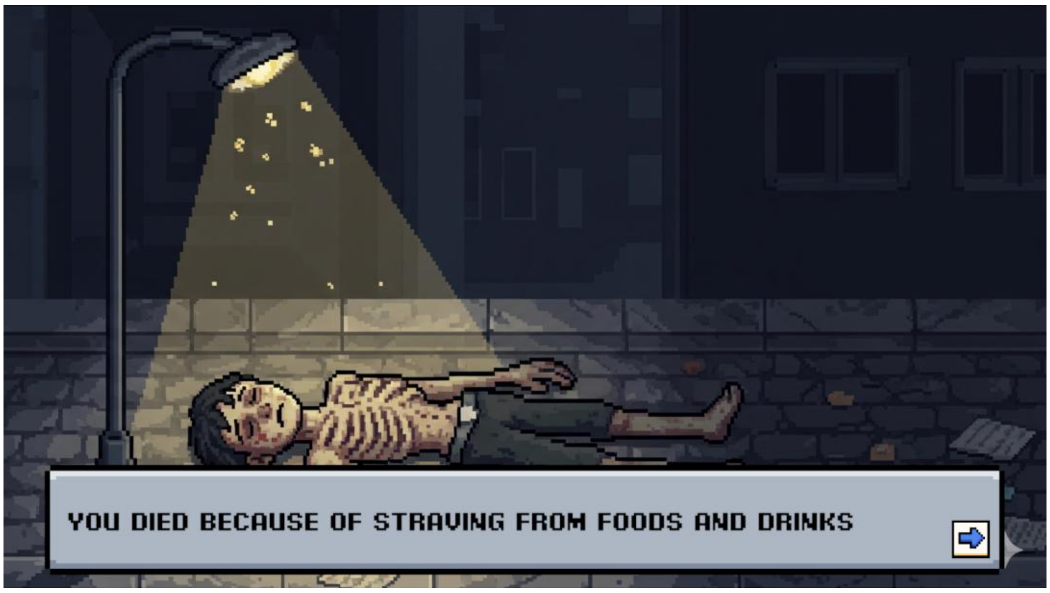
It guarantees investment profits

It allows you to buy luxury items immediately

It avoids paying taxes

Figure 4-2-20 Quiz System

Based on Figure 4-2-20, the quiz system which functioned as a feedback mechanism to evaluate how much financial knowledge the players had acquired throughout the gameplay. It was designed to simulate mid-term and final examinations commonly conducted in a university setting. The quiz system was set to two answering methods which were single-choice and multiple-choice questions. When players selected an incorrect answer, no marks were awarded whereas correct answers were rewarded marks with their overall score. The accumulated marks subsequently influenced the final ending of the game by linking academic assessments to the player in-game progression.



YOU DIED BECAUSE OF STRAVING FROM FOODS AND DRINKS

Figure 4-2-21 Multiple-ending System

A multiple-ending system was implemented to be displayed as the conclusion of the game which serving as an evaluation of the player's overall progression. The game simulated one university semester which players were expected to gain sufficient financial knowledge. The ending system examined both the player's financial status and their performance in the mid-term and final examinations. The endings were categorised into three main outcomes were becoming a fast-food worker, a company officer or a successful entrepreneur owned an entire city. The outcome depended on the player's examination grades and the amount of money accumulated by the end of the 90 days. If the examination grade was lower than a C or the player failed to attend the examinations, the fast-food worker ending was triggered. When the player did not achieve RM10,000 in savings and RM20,000 in total earnings but obtained results ranging from C to B, the company officer ending was triggered. Otherwise, if the player managed to save at least RM10,000 with total earnings of RM20,000 and secure an A grade, the successful entrepreneur ending was displayed. Apart from these endings, certain endings could be triggered in the midway of the gameplay. These included scenarios where the player's health or mental health dropped to zero or when bills were left unpaid beyond their due date.

4.3 Software Component Design

The software design of *Finance Journey in University* was implemented in Unity 6000.1.13f1 (LTS) using C# scripts as the programming language. Assets such as sprites and audio were integrated into the pipeline through Unity's editor and external tools such as Aseprite, Photoshop, and Adobe Audition were used. Majority of scripts in Unity were MonoBehaviour Script to deal with Unity functions.

The following subsections present the major software components:

- **Main Menu & Character Customization**
    - BodyPartsManager.cs - Handles the assignment and rendering of body part sprites for the player character.
    - BodyPartsSelector.cs - Provides a UI interface for selecting and customizing body parts.
    - MainMenu.cs - Manages the character naming process through the main menu.
    - PlayerData.cs (ScriptableObject) - Stores persistent character data such as name and selected body parts.
- **Player Movement and Interaction**

  - CharacterController2D.cs - Controls player movement, physics (Rigidbody2D), and animations.
  - CharacterInteract.cs - Handles player interaction with environment objects and NPCs.
  - TalkInteract.cs - Enables NPC dialogue initiation.
  - Actor.cs (ScriptableObject) - Defines NPC identity, including name and portrait sprite.
  - DialogueContainer.cs (ScriptableObject) - Stores dialogue lines and actor references.
  - NPCDefinition.cs (ScriptableObject) - Defines NPC-specific dialogue containers.
- **Dialogue System**
  - OptionDialogueTrigger.cs - Initiates branching dialogue choices.
  - OptionDialogueDefinition.cs (ScriptableObject) - Defines dialogue data including character portrait, question text, and response options.
  - DialogueActionHandler.cs - Processes dialogue outcomes and triggers associated actions.
- **Inventory System**
  - InventoryController.cs - Manages character inventory slots and item assignment.
  - InventoryPanel.cs - Displays the inventory UI for player interaction.
  - InventoryDetail.cs - Provides detailed item information within the UI.
  - ItemContainer.cs (ScriptableObject) - Defines item slot structures and references to item ScriptableObjects.
- **Store System**
  - ItemStorePanel.cs - Handles the store user interface when interacting with store clerks.
  - Trading.cs - Processes buy/sell transactions between the player and the store.
  - Store.cs - Represents the NPC store clerk and manages store availability.
- **Stock Market System**
  - StockMarket.cs - Implements stock asset logic, price changes, and market rules.
  - StockTrading.cs - Manages player interaction with stock assets (buying/selling).

    - ItemStockPanel.cs - Displays stock market data with up/down/no change indicators.
- **Bill System**
    - BillPanel.cs - User interface for displaying bills and expenses.
    - BillPanelController.cs - Handles billing operations within the player character's financial system.
- **Toolbar System**
    - ToolbarController.cs - Manages the quick-access toolbar for equipping and using items.
    - ItemToolbarPanel.cs - Displays toolbar UI.
    - ItemContainer.cs (reused) - Supports toolbar item references.
- **Time and Scheduling**
    - DayTime.cs - Controls the in-game clock, day/night cycle, and event timings.
    - TimetableManager.cs - Manages academic timetable for the player character.
    - AutomaticAttendanceTracker.cs - Automatically tracks attendance and updates attendance record.
- **Character Statistics**
    - Character.cs - Stores and manages player statistics such as health, finance, and relationships.
- **Achievement System**
    - AchievementManager.cs - Handles achievement tracking and presentation in the UI.
    - AchievementPopUpPanel.cs – Manage achievement pop up animation and UI for achievement pop up panel
- **Global Systems**
    - GameManager.cs - Centralized system managing global logic across scenes.
    - Transition.cs - Manages scene loading/unloading with persistence of core systems.
- **Data Storage**
    - ScriptableObjects - Used for structured and reusable game data (items, NPCs, dialogues).
    - PlayerPrefs - Stores lightweight data such as unlocked achievements.

Implementation of UI interface in Unity included elements such as panels were purposely to group and structure interface sections, while TextMeshPro and Text were utilised to present information with improved readability and visual quality. Images were integrated to represent icons, character portraits, and inventory items, contributing to a more immersive experience. Sliders were adopted for representing adjustable values, such as health or progress, and scrollbars in combination with scroll rects and masks were implemented to manage content lists, such as timetable panel and stock store panel made them more organised and readability.

4.4 System Components Interaction Operations

The system relied on close interaction among its system components where player actions and time-based events triggered responses across gameplay, interface and data management. This design ensured that all activities produced consistent and real-time outcomes. the Unity Input System transmitted player inputs to the CharacterController2D.cs which applied changes via Rigidbody2D and Animator for movement purpose. The Cinemachine package maintained camera focus and BoxCollider2D handled collisions and interactions. Financial transactions were managed through CharacterInteract.cs and ItemStorePanel.cs with Trading.cs updating the currency system and InventoryController.cs by added or deducted item from inventory. The stock market followed a similar flow with ItemStockPanel.cs, StockMarket.cs and StockTrading.cs to manage item prices, trade actions and currency synchronisation.

Dialogue interactions used TalkInteract.cs to retrieve lines from DialogueContainer.cs and NPCDefinition.cs while OptionDialogueTrigger.cs and DialogueActionHandler.cs handled choices and consequences in dialogue. Consumable items were managed by ToolbarController.cs and Character.cs to update health, hunger, thirst or mental health by reflecting them in the UI and reinforced with sound effects.

The DayTime.cs script managed time and triggered events such as hunger or thirst changes, timetable checks, attendance tracking and examination scheduling. It also initiated billing events where BillSystem.cs calculated the deadline of bill and validated payments.

Achievements were tracked via AchievementManager.cs which validated conditions such as earning RM10,000 or passing exams. A pop-up with DOTween animation and sound will be shown when achievements were achieved and the progress was stored using PlayerPrefs. Multiple endings were determined by the examination system, Character.cs status or progression logic in GameManager.cs which led to outcomes such as failure due to poor performance or health depletion.

Chapter 5
System Implementation

5.1 Hardware Setup

The hardware involved in this project was a personal computer, which supports the full process of game development. This includes asset creation, game design and implementation, animation, audio editing, and version control. The specification of the computer used were detailed below:

Description	**Specifications**
Model	Asus LAPTOP-EAU4SA7O
Processor	AMD Ryzen 5-3500U
Operating System	Windows 11
Graphic	AMD Radeon (TM) Vega 8 Graphics
Memory	12GB DDR4 RAM
Storage	475GB KINGSTON SSD

Table 5-1-1 Specifications of laptop

5.2 Software Setup

The project was developed using several software for game development. Game engine acted as the most important tool that helped developers to develop game in more efficient and effective way. There was some software used for modifying and creating assets that should be imported into Unity. Tool usage included:

- **Unity Hub 3.8.0 -** game engine hub which listed all project for game development
- **Unity Editor 6000.1.13f1 –** the game engine used for developing the entire game flow and logic, allowed developers to develop using pre-defined component and custom script which provided high flexibility
- **Aseprite v1.3.8.1** - used to create and modify pixel art assets like characters, objects, and tilemaps.
- **Adobe Photoshop 2020** - used to crop and edit images for better transparency and clarity.
- **Adobe Audition 2020** - used for editing and optimizing background music and sound effects.

- **GitHub** - used for version control, ensuring the project was backed up and manageable through iterative updates.

The project was developed using the Unity Editor version 6000.1.13f1 which was optimised for 2D game development. It provided numerous pre-defined components that helped developers to save time like RigidBody2D, Box Collider 2D and more. These components had completed functionality to handle the logic of game object where developers could work on custom script that required much time. UI package in Unity showed general UI that had already been made and popular to be used to speed up developers' development instead of sketching them from zero. The software Aseprite (v1.3.8.1) was utilised for designing, creating and modifying custom pixel art assets, including characters, objects and environment tiles. Some graphical resources were also acquired from online sources such as itch.io while Adobe Photoshop 2020 was occasionally used for background removal and image cropping tasks. Audio editing was done using Adobe Audition 2020, allowing for the creation and refinement of background music, sound effects and other audio elements. For version control and collaborative backup, a Git repository hosted on GitHub was used to track changes and store the project files securely. Most assets such as tilemaps, characters and buildings were in 16x16 and 32x32 pixel formats which were suitable for 2D top-down games.

5.3 Setting and Configuration

The project was positioned as a 2D top-down pixel art style of finance simulation games. The gameplay setting was set the resolution of the main scene in game was at 1920x1080 and the sprites were scaled consistently at 16x16, 32x32 or 64x64 pixels. The sprites pixels were set by its original pixel to match the actual size in the gameplay. Rendering pipeline used for *Finance Journey in University* was the Unity's default 2D rendering pipeline where suitable for a 2D top-down game. The camera system was configured with Cinemachine package by adding Cinemachine Brain component for main camera game object which followed Cinemachine Camera setting in virtual camera game object. The virtual camera set lens of camera to 6 which considered the distance between character with camera and position control was configured as followed player's character game object. Therefore, the camera will move followed by player's character movement smoothly with camera confiner component that blocked camera movement until it reached to the scene or map boundary. The input system was set like the other computer-based game application where WASD key was used for player movement such as going up, left, down and right while pressing shift key will make player's character moving in faster speed known as "run". In addition, the UI canvas render mode was in "Screen Space - Overlay" so the object in game unable to block the UI component in game like time panel, tool bar, etc. The canvas scaler for UI canvas had also configured the reference resolution to 1920x1080 to match the exact size of a PC screen. Background music and sound effect were playable throughout the audio manager system in Unity which acted like "ear" for player to listen to all audio clips in the game. Therefore, audio manager script was added into audio manager game object to activate the functionality of listening to all audio clips that had been added into the game scene.

5.4 System Operation

The project was categorised as 2D role-paying simulation game combined with life simulation and educational gameplay. It followed the player's finance journey during university life and stated the consequence of daily decisions and how it affected player's finance and status. The goal of *Finance Journey in University* was to let player experienced 90 days of semester in game while putting effort on planning personal finance, maintaining a good result in academic and taking care of character's status. Character status was the basic mechanic for players to survive in the game. The gameplay emphasised both academic success and financial literacy which demonstrating how these two aspects were interdependent in shaping long-term outcomes.

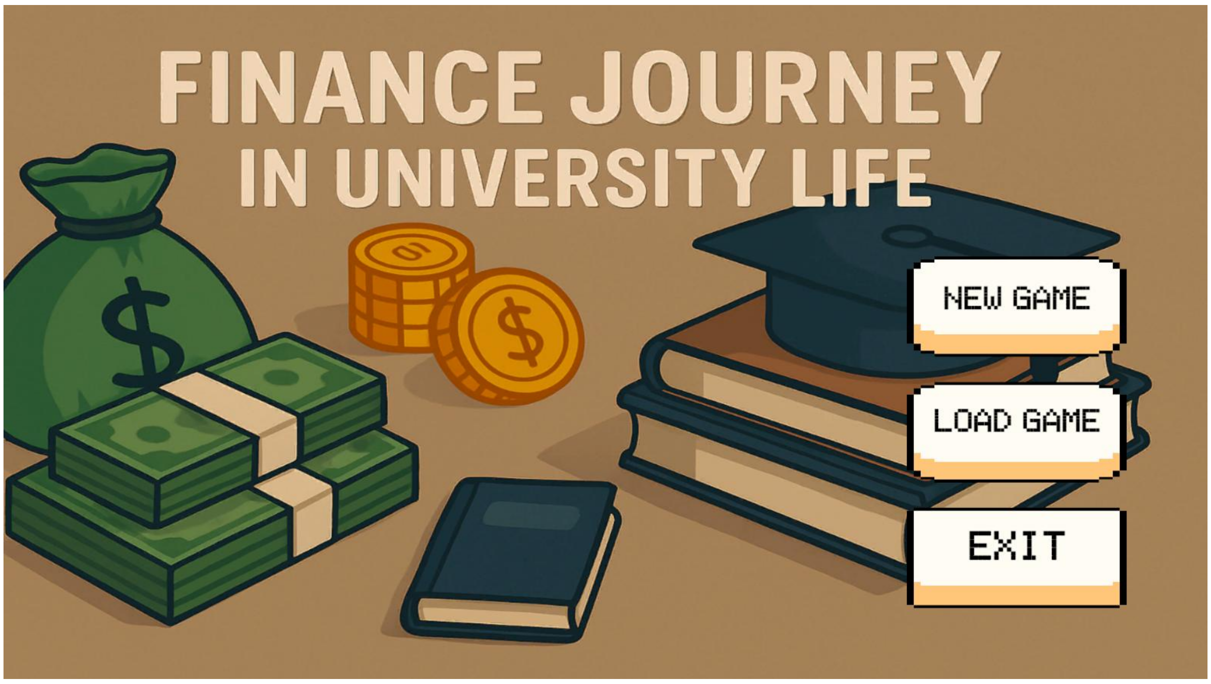


Figure 5-4-1 Main Menu Scene in *Finance Journey in University*

The game started with a main menu interface with buttons “New Game”, “Load Game” and “Exit” to perform action to start a new game, loading from existing saving process and quit from the application. If “New Game” button was selected, player would need to create a new character with customization to act as a university student in the game.

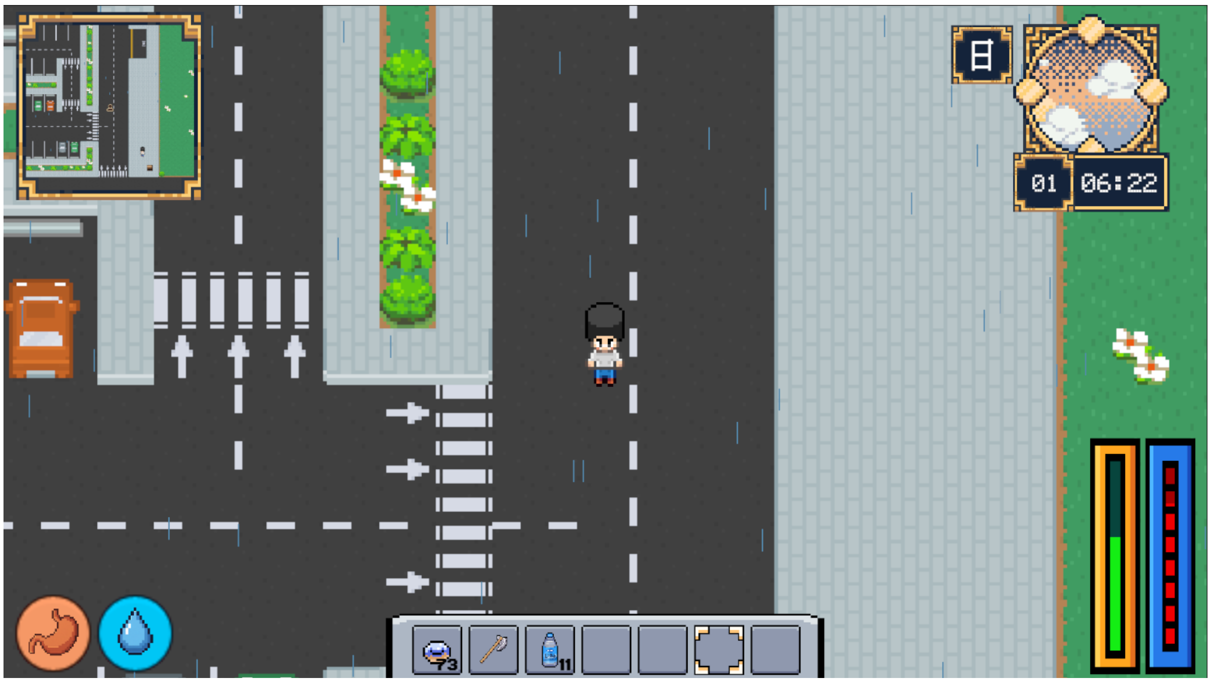


Figure 5-4-2 Starting Scene for Player

At the beginning of the game, the created character was placed on the exterior of the school building block called “Block N”. There were several UI that will keep showing in the whole gameplay which were mini map system sticked at the top left corner to show the larger view of whole scene in game and time system showed at the right top corner to show the entire progress in game. Character status was showed as the hunger and thirst value were sticked at left bottom corner with mental health and health status bar at right bottom corner. Then, the tool bar was placed at the middle bottom of the game and helped player to select the items and interact with them. The audio manager managed the audio played throughout the game and enhance player’s immersive experience. Throughout the gameplay, player will need to act as a student in a university and required to learn financial literacy knowledge while maintaining a good attendance in school and daily life aspect. In terms of player experience, the game aimed to provide a balance between entertainment and education.

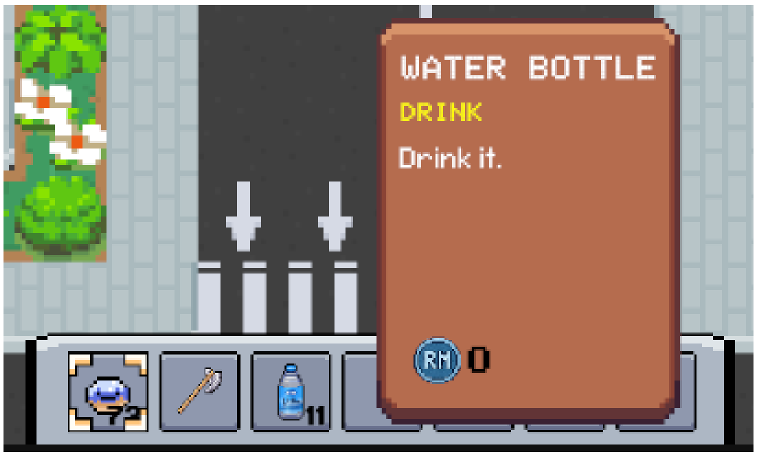


Figure 5-4-3 Tool Bar with Item Detail Panel

The highlighter in the tool bar showed the currently selected item, it helped player to clarify which item was selected and interact with it. When the mouse icon hover items, item detail panel will show and player could read the description and detail to know the effect of item after interacting with them.

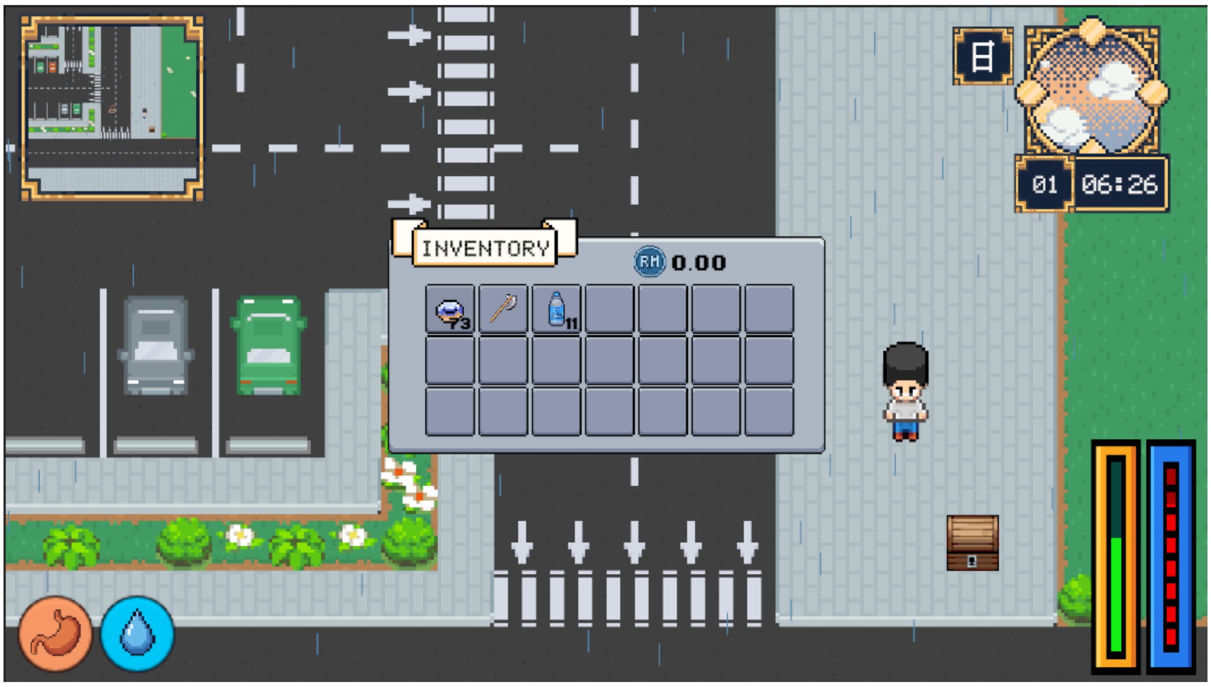


Figure 5-4-4 Opening Inventory System

Inventory system could be opened by pressing the 'E' key on the keyboard and could be close by pressing 'E' key again. With this system, player could check the existing items in player's bag and arrange it by clicking on item using mouse device. Some specific action performed like splitting the taken items into half quantity by right clicking on the item or taking the whole number of items by left clicking on it. Player could place the items into any slots by left-clicking the taken items on the empty slot otherwise the taken items will place and the items on the slot will be taken by mouse.

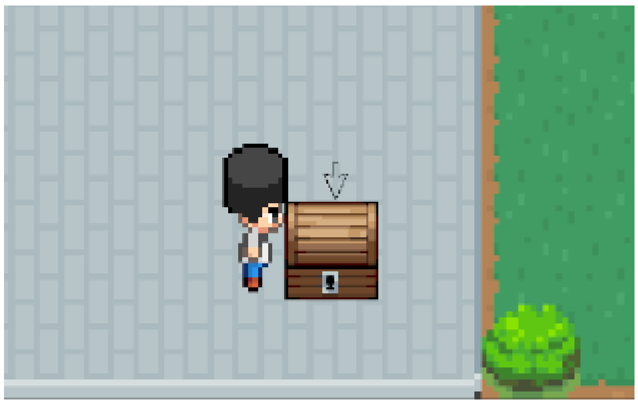


Figure 5-4-5 Interaction Feedback UI

By near to an interactable object, a black arrow will appear on top of the object and player's character could interact with it by right clicking on it. The interactable object could be an NPC, object or item in the tool bar with their own specific functionality.

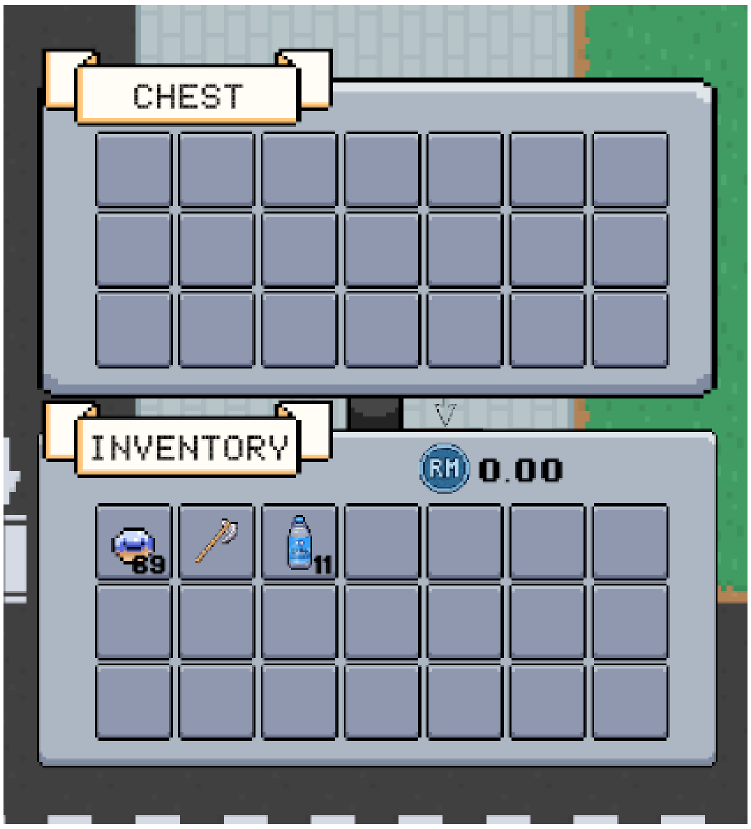


Figure 5-4-6 Opening Chest

For instance, a chest inventory interface will be show with inventory panel to place the items into chest panel. The inventory and chest item container will be saved in global so the data will be maintained while player transition through different scenes.

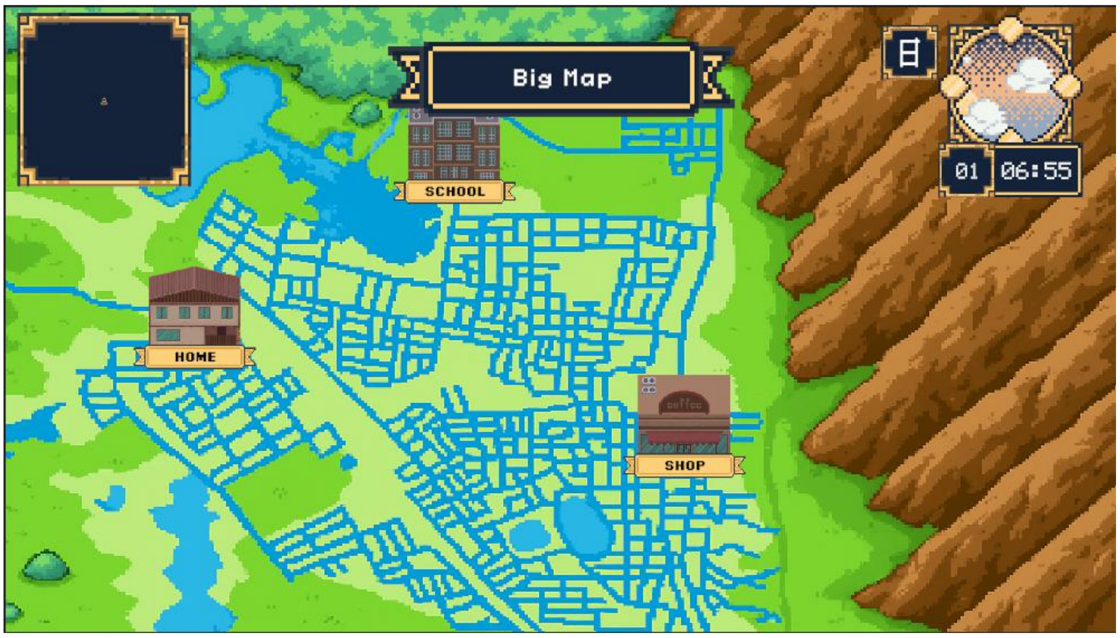


Figure 5-4-7 Big Map Transition

Movement of player between scenes was known as transition. There were several types of transition in the game, big map transition, scene transition and in-scene transition. Big map transition was to let player transition to place scenes where had longer distances compared to current place scene. Figure 5-4-7 presented Big Map scene where allowed player to left click on the "ribbon" that written the scene name and player will be brought to the following scene. Scene transition will load the next place scene from current scene while in-scene transition will just warp character to specific position on current place scene.

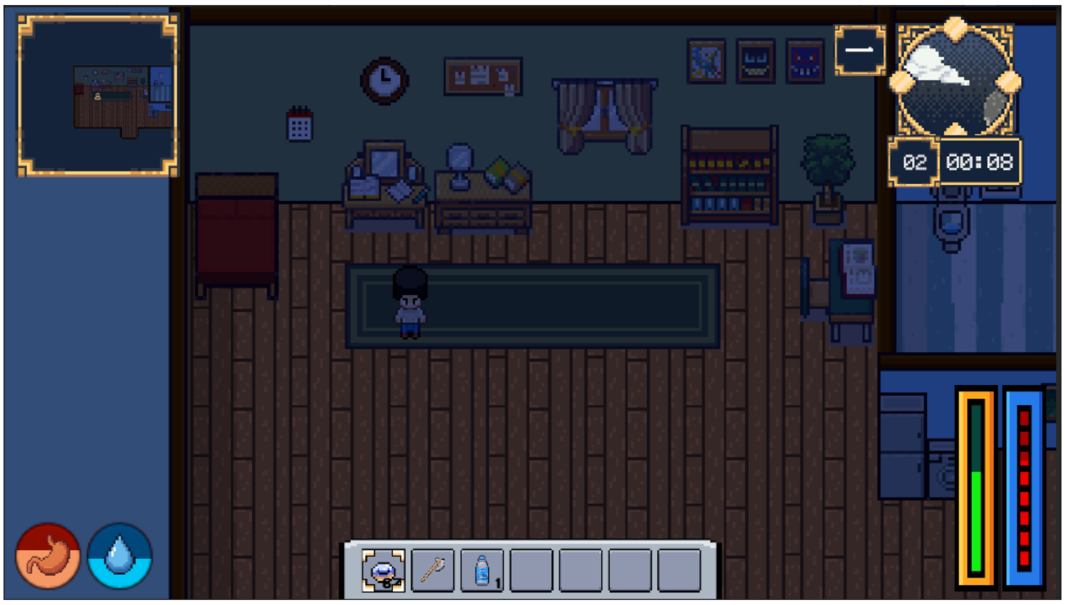


Figure 5-4-8 Character's Room at Night

The sprite of time system will be changed when time system reached specific time like 6 a.m., 12 p.m., 3 p.m., 5 p.m., 7 p.m. and 12 a.m. to show current phase of day. Character allowed to sleep in the bed by walking to the bed position of player's character room and the day will be passed to the next day 6 a.m. every night.

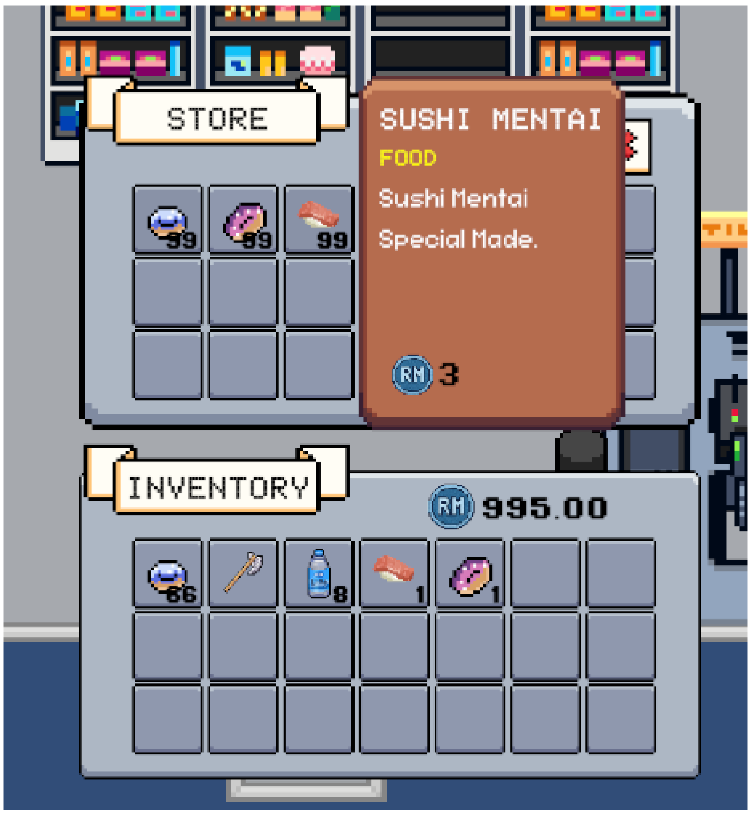


Figure 5-4-9 Buying Items from Store Market

The functionality of buying items like food and drinks on store market was important to maintain daily life for human as well as the character in the game. Player could check price and description by hovering the item to view the item detail panel. The RM icon with number shown in the inventory reflected current amount of money existing in character currency system. Item bought will directly added into character's inventory and amount of money deducted automatically.

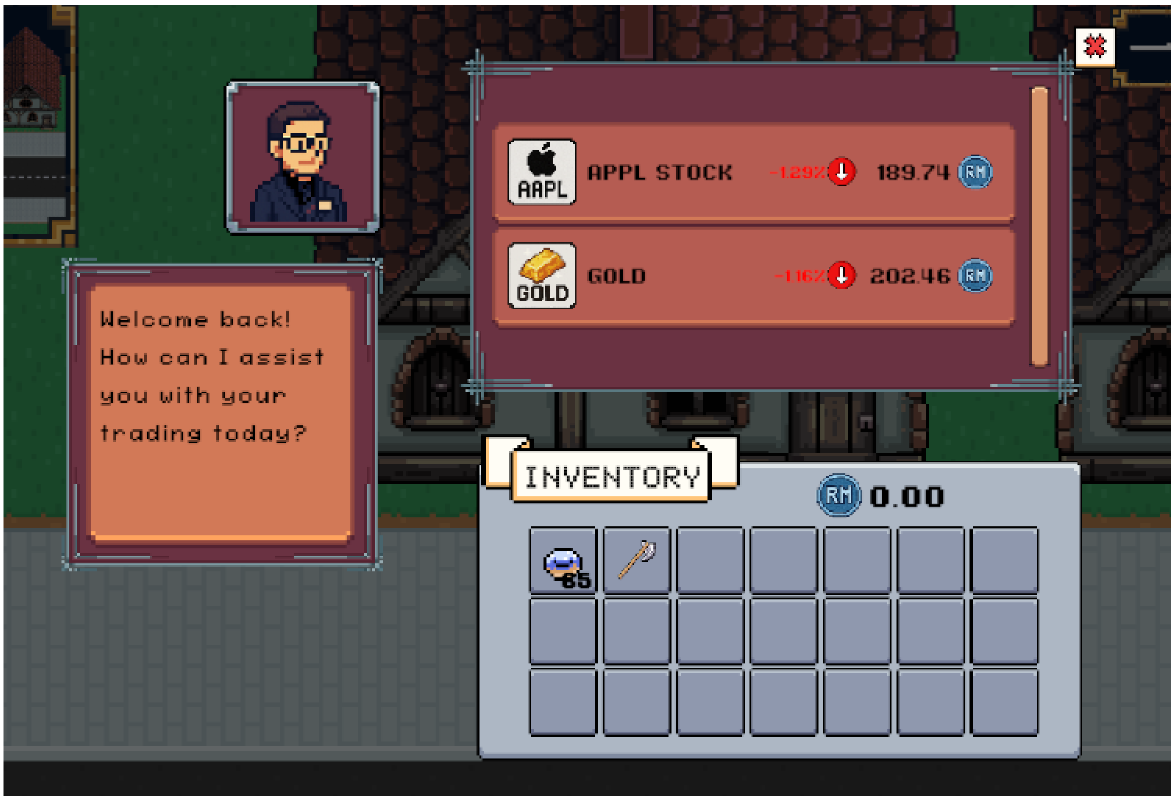
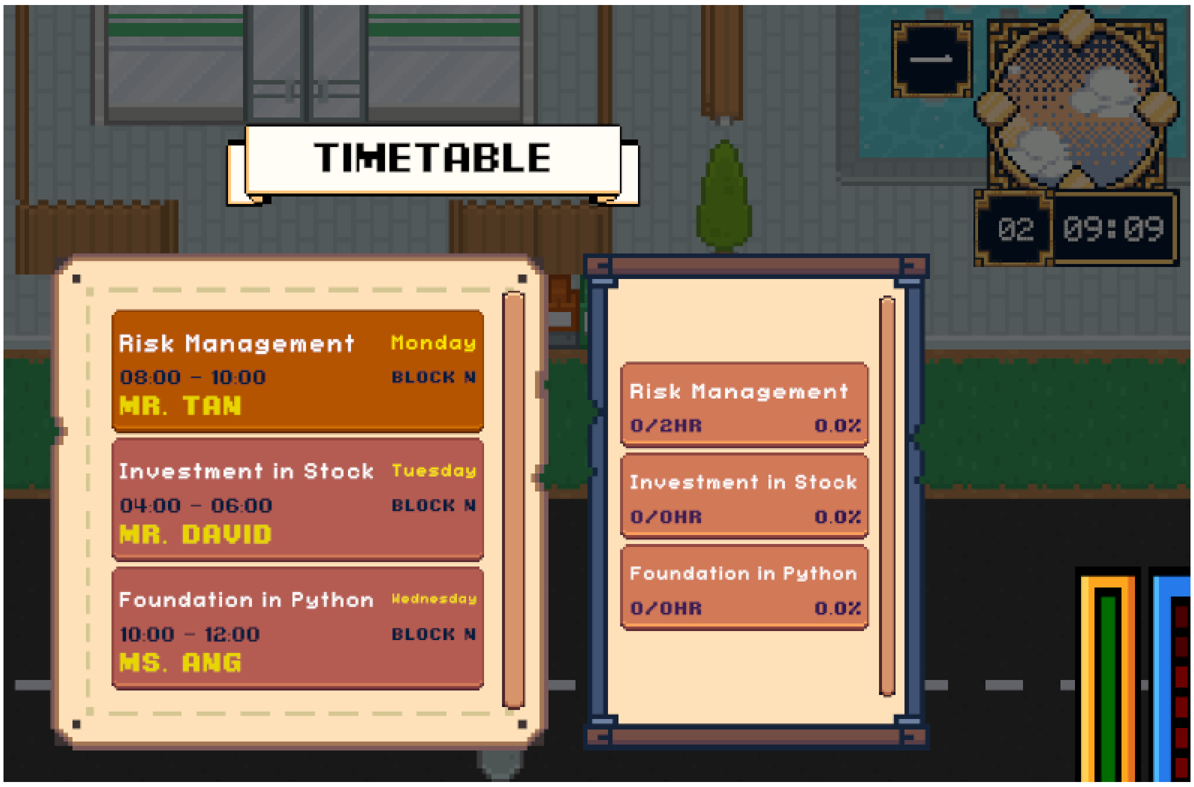

Welcome back! How can I assist you with your trading today?

Figure 5-4-10 Stock Market Fluctuation Price for Stock

The following figure showed the stock market system in *Finance Journey in University*, stock market panel show the detail of item directly instead of hovering on it. It was used to teach some financial knowledge and provide financial advice to player for exposing the shallow layer of stock market in real world. The price of stocks was changing over time to show the manipulation of stock in the real market and the risk of buying unrationed.


TIMETABLE

Risk Management Monday
08:00 - 10:00 BLOCK N
MR. TAN

Investment in Stock Tuesday
04:00 - 06:00 BLOCK N
MR. DAVID

Foundation in Python Wednesday
10:00 - 12:00 BLOCK N
MS. ANG

Risk Management
0/2HR 0.0%

Investment in Stock
0/0HR 0.0%

Foundation in Python
0/0HR 0.0%

02 09:09

Figure 5-4-11 Timetable UI when Class Time Reached

Moreover, player could press “T” key on keyboard to view the timetable panel to check the class should be attend in this semester. The timetable system was closely connected to time system which tracked the time in game and updated the class status and attendance of player’s character. For example, the slot of class changed slightly in colour to show player that the class was currently opening and player should attend the class just like how it showed in Figure 5-4-10. The achievement system also reflected the progress of player but it was optional because it was just a certificate to state player had completed some actions.

During the 45th day and 90th day of the game, player would need to take midterm and final examination just like a university student before the semester end. The panel show and player will be required to choose one of the answers for each question and the final score will be shown after the examination time. Lastly, player will be guided to the ending of the game which will state the outcome of player through academic and financial status to decide to different ending in the game.

5.5 Implementation Issues and Challenges

During the implementation of *Finance Journey in University*, several challenges were encountered which required careful consideration and adjustment throughout the development process.

One of the primary challenges was striking the right balance between the educational elements and the entertainment aspects of the game. The inclusion of financial literacy concepts such as budgeting, saving, and investing needed to be both accurate and meaningful, yet not so overwhelming that they diminished the enjoyment of gameplay. This balance was achieved through iterative testing and refinements, ensuring that financial lessons were integrated naturally into player decisions rather than being delivered in a rigid or purely instructional manner. Thus, another challenge was to ensure every system related to each other was functioning well and the logic was not crash to one another.

In addition, another significant challenge was in designing the market system. The aim was to simulate realistic price fluctuations for investment opportunities while keeping the system approachable for players who may not have prior financial knowledge. Early versions of the market system were either too simplistic which reduced educational value or too complex which discouraged player engagement. Therefore, Adjustments were made to introduce controlled fluctuations and simplified risk mechanics that maintained realism while supporting player understanding.

The game's visual presentation also posed challenges particularly in maintaining consistency across sprite assets. As the game made use of both externally sourced sprites and custom-designed ones created in Aseprite, discrepancies in style, resolution, and colour palettes initially created a disjointed visual appearance. Addressing this issue, sprites were carefully edited and standardised to ensure a natural pixel art style throughout the game environment.

5.6 Concluding Remark

The implementation of *Finance Journey in University* successfully brought together educational and entertainment elements into an immersive 2D top-down pixel art game. The project demonstrated how financial literacy concepts such as budgeting, saving, and investing could be embedded naturally into gameplay mechanics while maintaining player engagement. Although challenges were encountered in balancing complexity, refining the market system, and achieving visual consistency, these issues were resolved through iterative testing and adjustment.

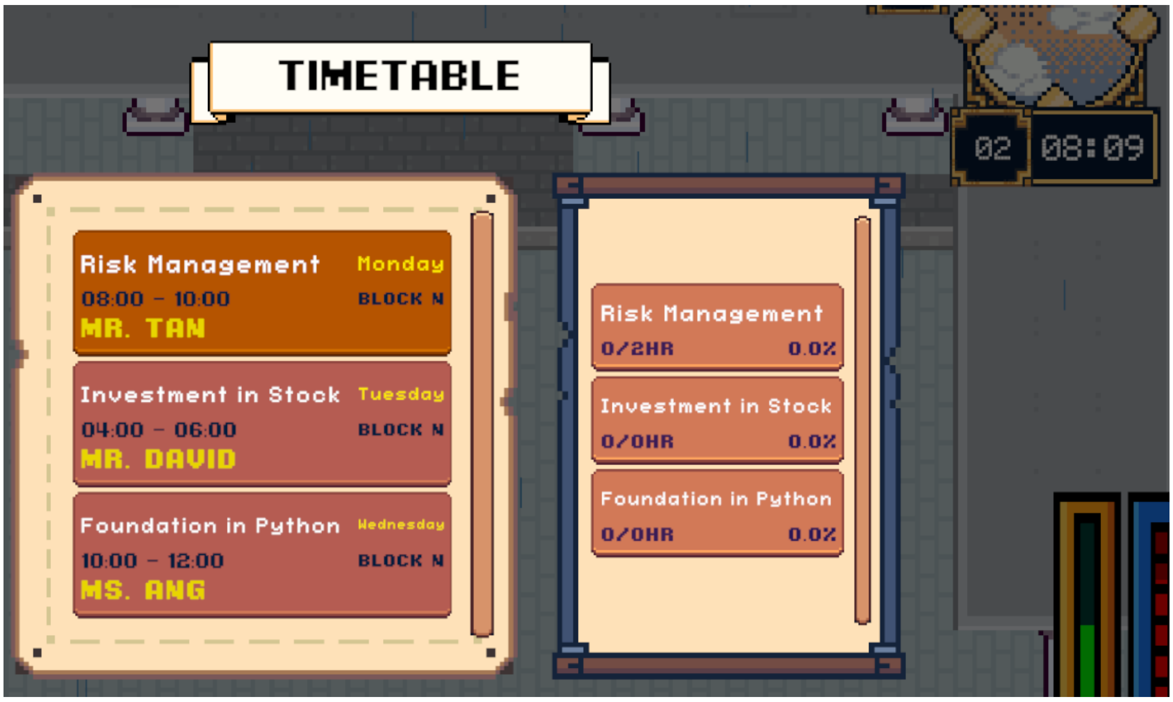
Overall, the implementation phase provided valuable insights into both the technical and design aspects of game development. The resulting system offered players an immersive and meaningful experience that reflected the realities of university life while teaching important financial principles. This stage of the project laid a strong foundation for future enhancements which ensuring that the game not only entertained but also achieved its educational objective of improving financial awareness among students.

Chapter 6

System Evaluation And Discussion

6.1 System Testing and Performance Metrics

The system testing of *Finance Journey in University* was carried out to ensure that the core functionalities of the game operated as intended and that the learning objectives were met. The testing focused on two major aspects were functional testing and performance metrics. Functional testing was performed to verify that all gameplay mechanics worked correctly in accordance with the design specifications. The character creation system allowed players to successfully customise their character, and movement was tested to confirm smooth control. Inventory management was checked by validating that a left mouse click enabled the player to pick up and move the entire stack of an item, while a right mouse click correctly split the stack in half. The placement of items into different inventory slots was also confirmed to work as expected. Sleep function was tested to ensure that it advanced the in-game clock to 6:00 a.m. on the following day.



Risk Management Monday
08:00 - 10:00 BLOCK N
MR. TAN

Investment in Stock Tuesday
04:00 - 06:00 BLOCK N
MR. DAVID

Foundation in Python Wednesday
10:00 - 12:00 BLOCK N
MS. ANG

Risk Management
0/2HR 0.0%

Investment in Stock
0/0HR 0.0%

Foundation in Python
0/0HR 0.0%

Figure 6-1-1 Attendance Percentage of 0 Before Attending to Class

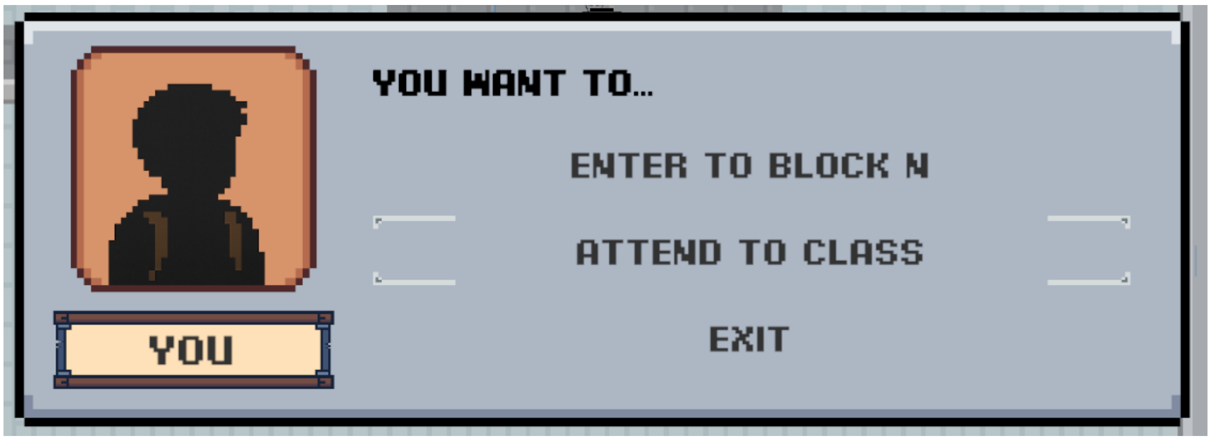
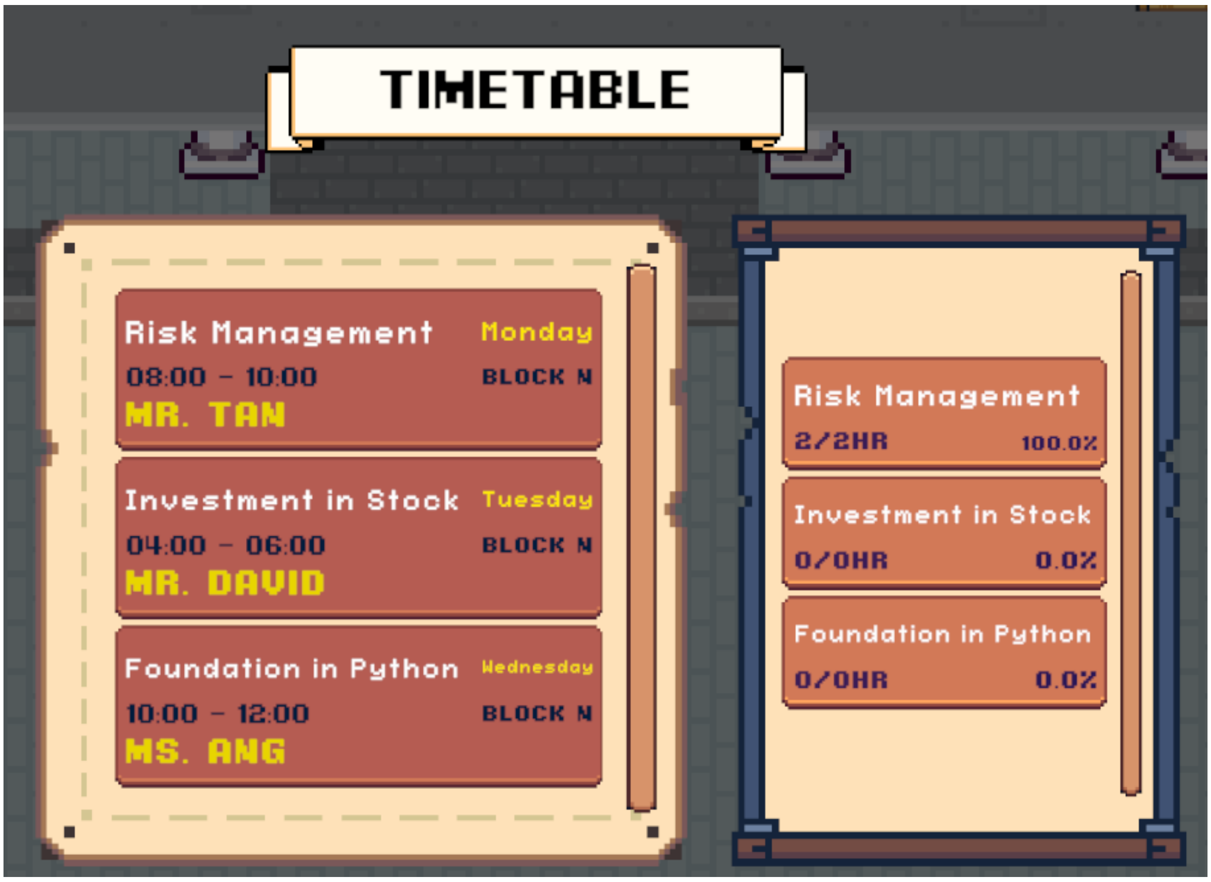


Figure 6-1-2 Option Dialogue for Attending to Class



Risk Management	Monday	
08:00 - 10:00	BLOCK N	
MR. TAN		
Investment in Stock	Tuesday	
04:00 - 06:00	BLOCK N	
MR. DAVID		
Foundation in Python	Wednesday	
10:00 - 12:00	BLOCK N	
MS. ANG		

Risk Management	
2/2HR	100.0%
Investment in Stock	
0/0HR	0.0%
Foundation in Python	
0/0HR	0.0%

Figure 6-1-3 Attendance Percentage of 100% After Attending to Class

The timetable and class attendance system were examined against the time system, confirming that attendance was recorded accurately. For example, attending a two-hour class resulted in a record of 2/2 hours (100%) in Figure 6-1-3, whereas missing the same class produced a record of 0/2 hours (0%) like Figure 6-1-1. Option dialogue system will appear when player walked to the entrance of university building “Block N” for selecting multiple actions to be performed like Figure 6-1-2.

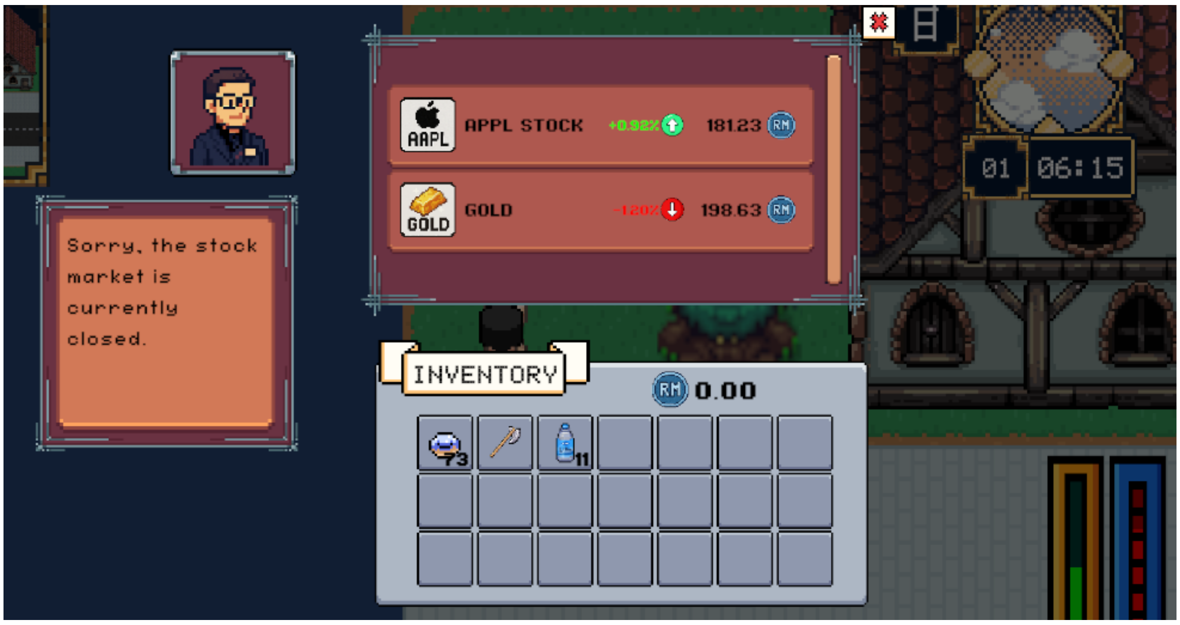

Sorry, the stock market is currently closed.
APPL STOCK +0.92% 181.23 RM
GOLD -1.20% 198.63 RM
INVENTORY
RM 0.00
01 06:15

Figure 6-1-4 Closed Market Dialogue When Stock Market Had Closed Market

The market system was tested in relation to the time system, confirming that shops opened and closed at the correct times. The purchasing process was validated to ensure that items could only be bought if the player had sufficient funds and if the store was open.

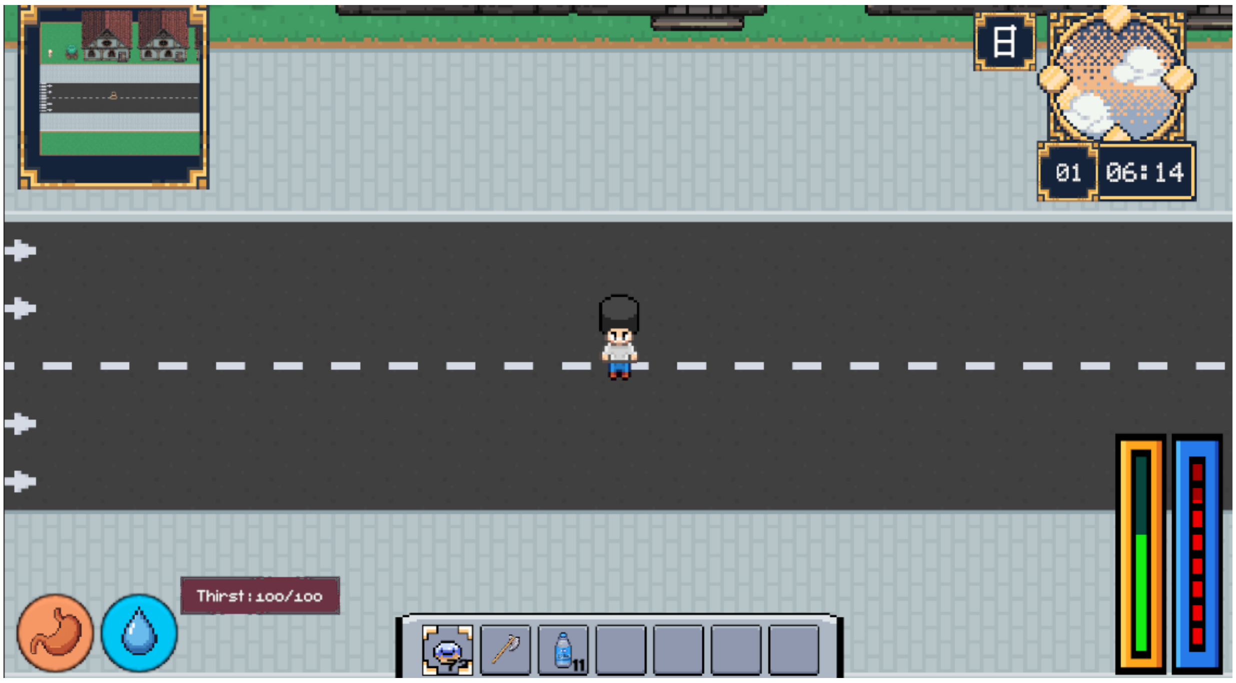


Figure 6-1-5 Character Status in Early Time

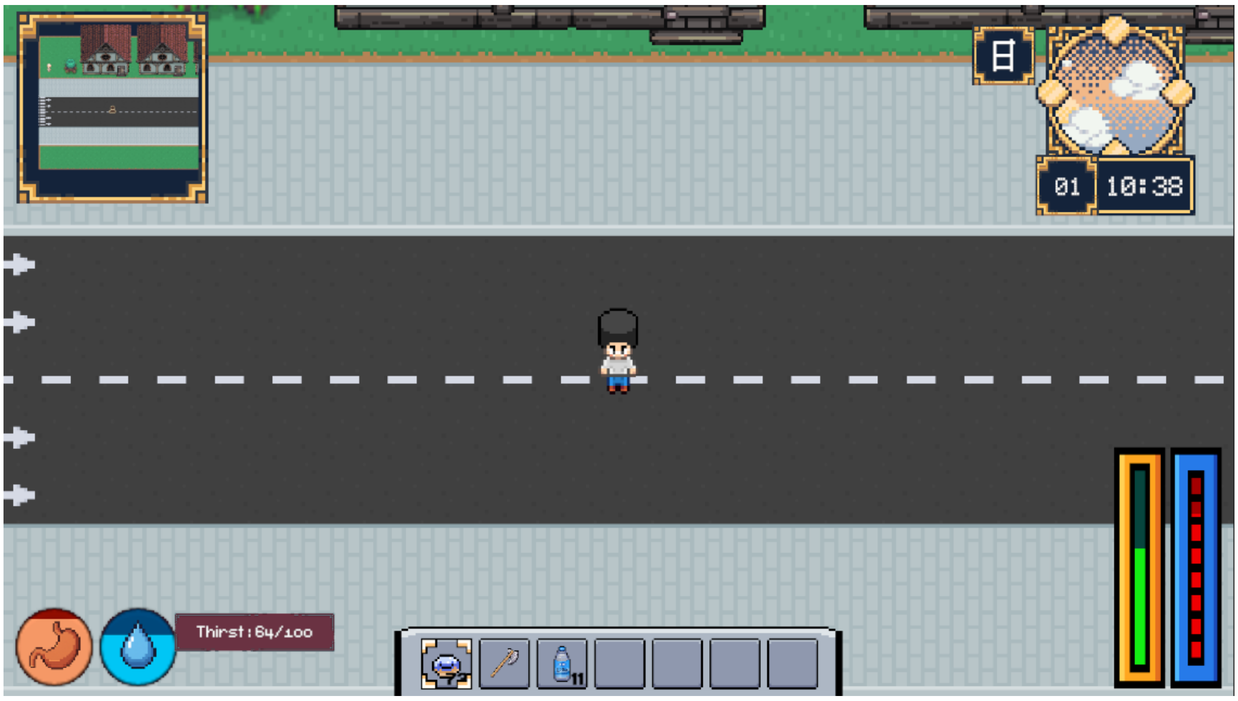


Figure 6-1-6 Character Status after 4 Hours

The status bars for hunger and thirst were observed to decrease progressively with time, while the health and mental health indicators were tested to ensure that when they reached zero, the character became uncontrollable and the game transitioned to the "Game Over" scene.

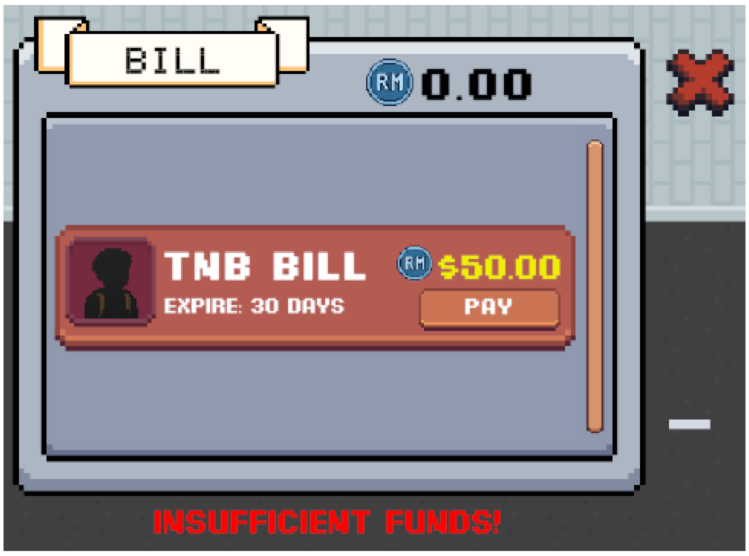
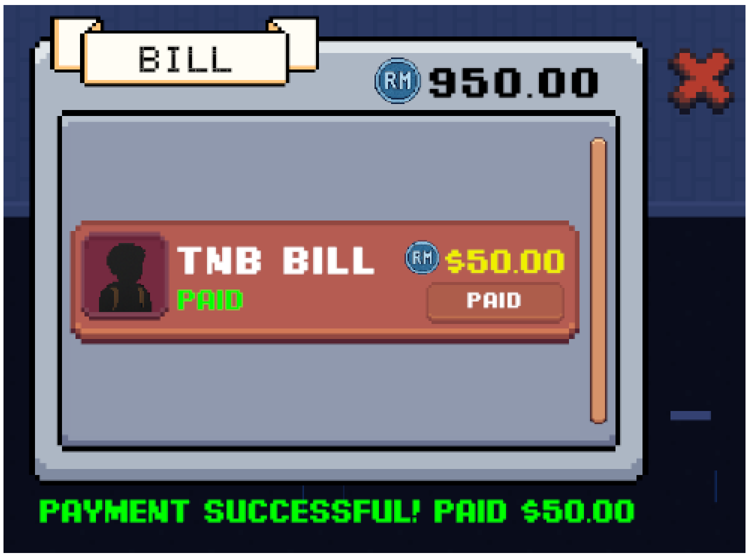


Figure 6-1-7 Failed Paying Bill



PAYMENT SUCCESSFUL! PAID $50.00

Figure 6-1-8 Paid Bill Successfully

The bill system was examined to confirm that payments were only successful when the player had sufficient funds and that bills were displayed as "Paid" once settled. When funds were insufficient, the error message will be shown. Furthermore, the player was unable to continue the game and redirected to the "Game Over" scene when any of the bill was expired.

The examination system was also tested to ensure that the player could only take the final examination if they maintained a minimum attendance level of 80%. Failure to attend the examination triggered an ending followed by the condition. The normal ending was tested to confirm that the game concluded after 90 in-game days and directed the player towards one of multiple possible endings.

From an educational perspective, performance was measured through the integration of examination system, achievement system and multiple ending paths. The examination system tested the player's theoretical knowledge of financial literacy including budgeting, saving and investment concepts. The achievement system encouraged players to apply financial strategies during gameplay by rewarding actions that reflected sound financial decisions. Finally, the multiple ending system provided different outcomes based on the player's academic results and financial situation which reflecting the depth of their understanding and application of personal finance. These elements together ensured the game was not only functioned correctly but also fulfilled its role as a financial education platform.

6.2 Testing Setup and Result

The testing of *Finance Journey in University* was conducted on a Windows-based PC using the Unity Editor build and run. The primary testing was performed by the developer through

direct play sessions in order to validate functionality and user experience. Each playthrough was designed to simulate a complete in-game semester of 90 days. Simulation this situation, the in-game time system was accelerated by applying a timescale of 100 in float variable. With each day consisting of 86,400 seconds and divided into phases of 900 seconds (15 minutes per phase), the system progressed through 96 phases per day, ensuring realistic time-based mechanics within a manageable testing duration. The testing methodology combined black box testing to validate the functionality of key features which user will be tested to assess gameplay experience and direct observation to identify performance issues or design inconsistencies.
Several issues were initially identified during the testing phase. These included the market not closing at the correct time, timetable inconsistencies with the time system, non-instantiation of scrollable item content, incorrect rendering layers that caused depth conflicts between objects and characters and occasional UI error messages. These issues were systematically addressed through debugging and code adjustments after the affected systems functioned correctly. Character movement, shop interactions, the examination system and the day/night cycle all performed as intended once the final testing ran for development. The market system opened and closed in line with the time system, and status bars such as hunger and thirst updated consistently with the passing of time. Attendance tracking was also validated to record accurately in relation to class participation.
From a performance standpoint, the game maintained a stable average frame rate of 60 FPS with no major crashes or freezes, even during extended sessions. Scene transitions and interactions remained smooth and reliable throughout testing. In terms of educational effectiveness, players who participated in user testing reported an improved awareness of financial concepts such as budgeting and saving. The examination and achievement systems further reinforced financial literacy by encouraging players to apply theoretical knowledge in practical, decision-based gameplay. Overall, the testing confirmed that the game achieved both its technical and educational objectives.

6.3 Project Challenges

The development of *Finance Journey in University* presented several challenges that required more time and effort to study with. One of the primary challenges was the overall scope of the project which needed deep research and development. Hence, more time was required to source and adapt sprite resources which familiarise with the Unity interface, development tools and implement the necessary coding for gameplay mechanics. The process of bug fixing was time-consuming as the integration of interconnected systems such as the inventory, time management, and market system required careful planning. Furthermore, iterative testing and the addition of extra features extended the workload as the game demonstrated significant potential for future development.

Another major challenge was required to integrate financial knowledge into the gameplay. Financial knowledges were unfamiliar to a computer science student and additional research was required to understand of personal finance principles such as budgeting, saving, and investment. This research was essential to combine theoretical financial knowledge into engaging and practical in-game scenarios. It ensured that the game was functioned not only as an entertainment application but also as an educational platform. On the technical side, challenges also encountered due to limited experience in handling Unity's object-oriented structure and the relationships between game objects and MonoBehaviour scripts. Bugs that occurred within the systems in game required additional time and effort to read and resolve.

6.4 Objectives Evaluation

The objective of developing a realistic and engaging 2D educational game that enhanced financial literacy among university and college students was achieved through the immersive life simulation mechanics of the game. Player was placed in the role of a university student, where they must manage not only academic responsibilities but also day-to-day financial decisions such as paying rent, handling bills, buying necessities, and saving for future needs. The top-down pixel art style and semester-based timeline created a relatable environment that mirrored the real-life experiences of young adults.

The game implemented objective of integrating essential concepts of personal finance through its finance system and decision-making mechanics. Players were required to budget their limited income between daily needs, social life, and long-term savings. The investment and market systems introduced fluctuating prices and risks, giving players the opportunity to practise strategies like DCA. By saving consistently, players also experienced the effects of compound interest over time which reinforcing the importance of long-term planning. These financial principles were not taught passively but were instead embedded within in-game challenges and choices to ensure player had applied them in relatable and student-focused contexts.

Lastly, the objective of motivating students to connect financial literacy with their personal aspirations and demonstrating how wealth management supports life goals was addressed by linking financial outcomes to personal achievements and game endings. Player who managed their finances effectively will gain more opportunities such as purchasing desired items, affording travel, or securing a better lifestyle while poor financial management restricts options and may even had a bad life. The game also tied academic results and financial discipline to long-term success which reflecting how both were essential to real-life aspirations. By embedding life goals into the reward structure, the game emphasized that financial planning was not an abstract skill but a tool for achieving dreams and securing future opportunities.

6.5 Concluding Remark

In conclusion, the evaluation of *Finance Journey in University* demonstrated that the project successfully reached its intended objectives as an educational and entertainment platform. Through testing process, the game was shown to operate normally with stable performance on PC device. On the other hand, the challenges encountered during development provided lessons in game development, critical thinking and problem-solving skills which enhanced the result of the work. By combining realistic university life simulation with personal finance education, the game offered an engaging and meaningful experience that encouraged players to apply financial knowledge in decision-making in real life. Lastly, the project mentioned the potential of game application as a tool to enhance financial awareness among students alongside with entertainment value.

Chapter 7
Conclusion And Recommendation

7.1 Conclusion

The development of the project *Finance Journey in University* had successfully demonstrated how a digital game could be used as an educational tool to enhance financial literacy among university students. The project aimed to bridge the gap between theoretical financial concepts and practical by embedding real-life scenarios into an engaging interactive game application. Throughout the gameplay of the application, players were able to experience the consequences of their financial decisions in a controlled and realistic logic environment.

The system integrated multiple components such as the time cycle, inventory management, market simulation, bill payment system, examinations, and achievement tracking. Each of these elements worked simultaneously to provide a real situation of the challenges students often faced during their way in academic journey. The project reinforced the importance of balance in personal life compared to only focusing on personal wealth by adding features character status which simulating student's health, mental health, hunger and even thirst. This approach was used to increase immersive and difficulty in the project. The project also achieved its educational objectives by embedding financial concepts such as Dollar-Cost Averaging (DCA), savings strategies, and budgeting within the game mechanics. Students were encouraged to learn through practical rather than just memorised financial theory.

Nevertheless, certain limitations were identified. The game's development was constrained due to time and resource restrictions. It limited the variety of scenarios and the complexity of the financial system implemented in the project. Additionally, the system performed correctly under normal conditions on PC device which limited the platform for playing the game application. These restrictions provided chance for future refinement and development if the project was able to update and operate regularly.

In conclusion, the project had met its objectives by producing an interactive, educational and entertaining tool that solved the financial challenges faced by university students. It illustrated the abilities of using game-based learning to deliver practical knowledge and skills and build a foundation for further research and development in this area.

7.2 Recommendation

Based on the experience and outcomes of this project, several recommendations could be made for future work and improvement. Firstly, the game could include more financial scenarios and knowledge in the project. The current currency system only focused on fundamental aspects such as savings, investments, and bill management. It ignored more advanced concepts such as credit management and loans that would provide players with a deeper understanding of financial literacy.

Secondly, improvements to the user experience should be considered. Although the game was functional and visually consistent, more animation could be included like eat, drink and interact in the project. The facial expression of character and NPC could be added into the project to make players had more immersive experience. Same as the achievements system, more achievements could be added into the system and rewards were given to motivate players for completing achievements aggressively.

Thirdly, integration of multiplayer or collaborative features could significantly increase engagement and fun. players were allowed to interact, trade, or compete with each another would make social relationship and economic more realistically for the project. This improvement would also encourage teamwork and discussions about financial decision-making among peers thus increase the educational effect.

Furthermore, the project could be promoted broader by making it cross-platform like mobile device which was popular for current generation. As students increasingly relied on mobile devices for learning and entertainment, it provided accessibility across multiple platforms would allow the game to reach a wider target audience. This would require adjustments in performance, interface scaling, storage efficiency and implementation techniques.

It was also recommended that the game should undergo more evaluation through testing with a larger sample of university students as the project's target audience. Gathering quantitative and qualitative feedback from different university students' groups would provide deeper insights about the game's strengths and weaknesses so improvement could be implemented to make the project presented in better performance.

Finally, the game could be enhanced by communicating with financial education programmes or institutional support. By integrating the system into workshops or curriculum activity events, universities could provide students with a practical and interactive method of reinforcing their financial skills.

In summary, the project had successfully established the framework for the valuable educational tool with development, integration and refinement frequently. It could become a powerful tool for educating financial education at universities or colleges to change students' life from struggling with poor financial planning in life.

REFERENCES

[1] A. Barczak and H. Woźniak, “Comparative Study on Game Engines,” *Studia Informatica*, no. 23, pp. 5–24, Dec. 2020, doi: https://doi.org/10.34739/si.2019.23.01.

[2] A. Caroline, G. Potrich, K. Mendes Vieira, W. Mendes-Da-, S. Fundação, and G. Vargas, ‘Article in Management Research Review’, 2016, doi: 10.1108/MMR-06-2014-0143.

[3] A. Lusardi and O. S. Mitchell, ‘The Importance of Financial Literacy: Opening a New Field’, *Journal of Economic Perspectives*, vol. 37, no. 4, pp. 137–154, 2023, doi: 10.1257/jep.37.4.137.

[4] B. S. Benti and G. Stadtmann, ‘Animal Crossing: New Horizons meets “Maslow’s pyramid”’, *Hum Behav Emerg Technol*, vol. 3, no. 5, pp. 1172–1179, Dec. 2021, doi: 10.1002/hbe2.288

[5] B. S. Benti and G. Stadtmann, ‘B|Orders in Motion in the Video Game Industry: An Analysis Based on Animal Crossing: New Horizons’, *Hum Behav Emerg Technol*, vol. 2022, 2022, doi: 10.1155/2022/4452900.

[6] C. Liu, T. Franklin, R. Shelor, J. Reuter, and S. Moriarty, ‘A Learning Game For Youth Financial Literacy Education In The Teen Grid Of Second Life Three-Dimensional Virtual Environment’, 2011.

[7] D. Handarini, H. Mulyani, and N. Susilowati, ‘The Application of Monopoly Game Learning Media in Increasing Student Motivation in Learning Basic of Accounting in Vocational School’, 2022.

[8] D. Ibrahim, R. Harun, and Z. Mohamed Isa, ‘A Study on Financial Literacy of Malaysian Degree Students UNE ÉTUDE SUR LES CONNAISSANCES FINANCIÈRES DES ÉTUDIANTS MALAISIENS’, 2009.

[9] G.Carlos and H. Adinovam, “COMPARATIVE STUDY OF GAME ENGINES,”, vol. 8, no. 8, pp. 205-224, Jun. 2019 *Fatece.edu.br*, 2025. Available: https://fatece.edu.br

[10] H. Lim, S. J. Heckman, J. C. Letkiewicz, and C. P. Montalto, ‘Financial Stress, Self-Efficacy, and Financial Help-Seeking Behavior of College Students’, 2014. [Online]. Available: http://ssrn.com/abstract=2537579

[11] I. Azer and S. A. Mohamad, ‘Exploring Financial Management Practices and Problems among Students’, *International Journal of Academic Research in Business and Social Sciences*, vol. 8, no. 12, Dec. 2018, doi: 10.6007/ijarbss/v8-i12/5762.

[12] "Introduction to Godot," *Godot Engine documentation*, 2025. Available: https://docs.godotengine.org/en/4.4/getting_started/introduction/introduction_to_godot.html#what-was-godot (accessed Sep. 11, 2025).

[13] K. Ergün, 'Financial literacy among university students: A study in eight European countries', *Int J Consum Stud*, vol. 42, no. 1, pp. 2–15, Jan. 2018, doi: 10.1111/ijcs.12408.

[14] K. Wiboonsin and W. Kasemsukpipat, 'Wealth Creation: Serious Game Development to Improve Financial Literacy for High School Students'.

[15] M. R. D. Prihartono and N. Asandimitra, 'Analysis Factors Influencing Financial Management Behaviour', *International Journal of Academic Research in Business and Social Sciences*, vol. 8, no. 8, Aug. 2018, doi: 10.6007/ijarbss/v8-i8/4471.

[16] Ö. İSKENDER, 'Identification with Game Characters: Theoretical Explanations, Predictors, and Psychological Outcomes', *Psikiyatride Güncel Yaklaşımlar*, vol. 15, no. 2, pp. 203–219, Jun. 2023, doi: 10.18863/pgy.1104693.

[17] P. Hourdequin, E. Bostelmann, J. Dehaan, D. M. Jones, and F. Poole, 'Social learning and literacy affordances in Animal Crossing: New Horizons Item Details Key points'. [Online]. Available: https://www.llpjournal.org/2020/09/02/hourdequin-social-learning-with-animal-crossing-new-horizons.html

[18] Ramesha V, 'APPLICATION OF MASLOW'S THEORY IN FINANCIAL PLANNING'.

[19] R. Salama and M. Elsayed, "A live comparison between Unity and Unreal game engines," *Global Journal of Information Technology: Emerging Technologies*, vol. 11, no. 1, pp. 01–07, Apr. 2021, doi: https://doi.org/10.18844/gjit.v11i1.5288.

[20] S. B. Shanklin and C. R. Ehlen, "Using The Monopoly Board Game As An Efficient Tool In Introductory Financial Accounting Instruction," *Journal of Business Case Studies (JBCS)*, vol. 3, no. 3, pp. 17–22, Jul. 2007, doi: https://doi.org/10.19030/jbcs.v3i3.4852.

[21] S. Heckman, H. Lim, and C. Montalto, 'Factors Related to Financial Stress among College Students', *Journal of Financial Therapy*, vol. 5, no. 1, Aug. 2014, doi: 10.4148/1944-9771.1063.

[22] S. L. Britt, A. Canale, F. Fernatt, K. Stutz, and R. Tibbetts, 'Financial Stress and Financial Counseling: Helping College Students', 2015.

[23] S. L. Britt, M. R. Mendiola, G. H. Schink, R. H. Tibbetts, and S. H. Jones, 'Financial Stress, Coping Strategy, and Academic Achievement of College Students', *Journal of Financial Counseling and Planning*, vol. 27, no. 2, pp. 172–183, 2016, doi: 10.1891/1052-3073.27.2.172.

[24] T. Jappelli and M. Padula, “Investment in financial literacy and saving decisions,” *Journal of Banking & Finance*, vol. 37, no. 8, pp. 2779–2792, Aug. 2013, doi: https://doi.org/10.1016/j.jbankfin.2013.03.019.

[25] “Unreal Editor Interface Overview | Community tutorial,” *Epic Games Developer*, 2025. Available: https://dev.epicgames.com/community/learning/tutorials/PnXj/unreal-engine-unreal-editor-interface-overview

[26] U. Technologies, “Unity - Manual: Unity User Manual (2019.3),” *docs.unity3d.com*. Available: https://docs.unity3d.com

[27] V. VEELEN Darren, ‘Teaching Basic Financial Literacy Using the Monopoly™ Board Game in English as a Foreign Language Education’.

APPENDICES

C# Script Code:

Item.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(menuName = "Data/Item")]
public class Item : ScriptableObject
{
   public enum ItemType
   {
      Food,
      Drink,
      Item,
      Tool,
      Stock
   }

   public string Name;
   public bool stackable;
   [TextArea(3, 5)]
   public string description;
   public ItemType itemType = ItemType.Item;
   public Sprite icon;
   public ToolAction onAction;
   public ToolAction onTileMapAction;
   public ToolAction onItemUsed;
   public bool itemIconHighlight;
   public GameObject itemPrefab;
   public int price;
   public bool canBeSold = true;

   [Header("Consumable Properties")]
   public bool consumable;
   [SerializeField] int hungerRestoreAmount = 10;
   [SerializeField] int thirstRestoreAmount = 0; // New thirst restore property
   [SerializeField] int healthRestoreAmount = 0;
   [SerializeField] int happinessRestoreAmount = 0;

   // Public getters for the restore amounts
   public int HungerRestoreAmount => hungerRestoreAmount;
   public int ThirstRestoreAmount => thirstRestoreAmount; // New getter
   public int HealthRestoreAmount => healthRestoreAmount;
   public int HappinessRestoreAmount => happinessRestoreAmount;
}
```

ItemContainer.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[Serializable]
public class ItemSlot
{
    public Item item;
    public int count;

    public void Copy(ItemSlot slot)
    {
        item = slot.item;
        count = slot.count;
    }

    public void Set(Item item, int count)
    {
        this.item = item;
        this.count = count;
    }

    public void Clear()
    {
        item = null;
        count = 0;
    }
}

[CreateAssetMenu(menuName = "Data/Item Container")]
public class ItemContainer : ScriptableObject
{
    public List<ItemSlot> slots;
    public bool isDirty;

    public void Add(Item item, int count = 1)
    {
        isDirty = true;

        if (item.stackable == true)
        {
            ItemSlot itemSlot = slots.Find(x => x.item == item);
            if (itemSlot != null)
            {
                itemSlot.count += count;
            }
            else
            {
                itemSlot = slots.Find(x => x.item == null);
                if (itemSlot != null)
                {
                    itemSlot.item = item;
                    itemSlot.count = count;
                }
            }
        }
        else
        {
            //add non stackable item to ours item container
            ItemSlot itemSlot = slots.Find(x => x.item == null);
            if (itemSlot != null)
            {
                itemSlot.item = item;
            }
        }
    }

    public void Remove(Item itemToRemove, int count = 1)
    {
        isDirty = true;

        if (itemToRemove.stackable)
        {
            ItemSlot itemSlot = slots.Find(x => x.item == itemToRemove);
            if (itemSlot == null)
            {
                return;
            }

            itemSlot.count -= count;
            if (itemSlot.count <= 0)
            {
                itemSlot.Clear();
            }
        }
        else
        {
            while (count > 0)
            {
                count -= 1;

                ItemSlot itemSlot = slots.Find(x => x.item == itemToRemove);
                if (itemSlot == null)
                {
                    break;
                }
                itemSlot.Clear();
            }
        }
    }

    internal bool FreeSpace()
    {
        for (int i = 0; i < slots.Count; i++)
        {
            if (slots[i].item == null)
            {
                return true;
            }
        }
        return false;
    }

    internal bool CheckItem(ItemSlot checkingItem)
    {
        ItemSlot itemSlot = slots.Find(x => x.item == checkingItem.item);

        if (itemSlot == null)
        {
            return false;
        }

        if (checkingItem.item.stackable)
        {
            return itemSlot.count > checkingItem.count;
        }

        return true;
    }
}
```

InventoryPanel.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InventoryPanel : ItemPanel
{
    private RectTransform rectTransform;
    private float normalXPosition = 0f;
    private float normalYPosition = 0f;
    private float twoPanelYPosition = -160f;
    private float StockPanelXPosition = 170f;
    private float StockPanelYPosition = -190f;


    private void Awake()
    {
        rectTransform = GetComponent<RectTransform>();
    }

    public override void OnClick(int id)
    {
        GameManager.instance.dragAndDropController.OnClick(inventory.slots[id]);
        Show();
    }

    public void SetNormalPosition()
    {
        if (rectTransform != null)
        {
            Vector2 currentPosition = rectTransform.anchoredPosition;
            rectTransform.anchoredPosition = new Vector2(normalXPosition, normalYPosition);
        }
    }

    public void SetTwoPanelPosition()
    {
        if (rectTransform != null)
        {
            Vector2 currentPosition = rectTransform.anchoredPosition;
            rectTransform.anchoredPosition = new Vector2(normalXPosition,
twoPanelYPosition);
        }
    }

    public void SetStockPanelPosition()
    {
        if (rectTransform != null)
        {
            Vector2 currentPosition = rectTransform.anchoredPosition;
            rectTransform.anchoredPosition = new Vector2(StockPanelXPosition,
StockPanelYPosition);
        }
    }
}
```

InventoryButton.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;
using static UnityEditor.Progress;

public class InventoryButton : MonoBehaviour, IPointerClickHandler, IPointerEnterHandler,
IPointerExitHandler
{
    [SerializeField] Image icon;
    [SerializeField] Text text;
    [SerializeField] Image highlight;
    int myIndex;
    ItemPanel itemPanel;
    ItemSlot currentSlot;
    private ItemDetailPanel itemDetailPanel;

    public void SetIndex(int index)
    {
        myIndex = index;
    }

    public void SetItemPanel(ItemPanel source)
    {
        itemPanel = source;
    }

    public void SetDetailPanel(ItemDetailPanel panel)
    {
        itemDetailPanel = panel;
    }

    public virtual void Set(ItemSlot slot)
    {
        currentSlot = slot;
        icon.gameObject.SetActive(true);
        icon.sprite = slot.item.icon;
        if (slot.item.stackable == true)
        {
            text.gameObject.SetActive(true);
            text.text = slot.count.ToString();
        }
        else
        {
            text.gameObject.SetActive(false);
        }
    }

    public virtual void Clean()
    {
        currentSlot = null;
        icon.sprite = null;
        icon.gameObject.SetActive(false);
        text.gameObject.SetActive(false);
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        // Check if it's left click (0) or right click (1)
        if (eventData.button == PointerEventData.InputButton.Left)
        {
            // Left click behavior
            itemPanel.OnLeftClick(myIndex);
        }
        else if (eventData.button == PointerEventData.InputButton.Right)
        {
            // Right click behavior
            itemPanel.OnRightClick(myIndex);
        }
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        if (itemPanel != null && itemPanel.ShowItemDetails && currentSlot != null &&
currentSlot.item != null)
        {
            itemDetailPanel.ShowItemDetails(currentSlot.item, Input.mousePosition);
        }
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        if (itemDetailPanel != null)
        {
            itemDetailPanel.Hide();
        }
    }

    public void Highlight(bool b)
    {
        highlight.gameObject.SetActive(b);
    }
}
```

Character.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[Serializable]
```

```
public class Stat
{
   public int maxVal;
   public int currVal;

   public Stat(int max, int curr)
   {
      maxVal = max;
      currVal = curr;
   }

   internal void Subtract(int amount)
   {
      currVal -= amount;
      if (currVal < 0)
      {
         currVal = 0; // Prevent negative values
      }
   }

   internal void Addition(int amount)
   {
      currVal += amount;
      if (currVal > maxVal)
      {
         currVal = maxVal;
      }
   }

   internal void SetToMax()
   {
      currVal = maxVal;
   }

   internal bool IsEmpty()
   {
      return currVal <= 0;
   }

   internal bool IsFull()
   {
      return currVal >= maxVal;
   }
}

public class Character : MonoBehaviour
{
   [Header("Character Stats")]
   public Stat Health;
```

```
    public Stat Happiness;
    public Stat Hunger;
    public Stat Thirst;

    [Header("Stat Deduction Settings")]
    [SerializeField] private int hungerDeductionAmount = 1;
    [SerializeField] private int thirstDeductionAmount = 1;
    [SerializeField] private int hungerDeductionInterval = 3; // Every 3 time ticks (45 minutes)
    [SerializeField] private int thirstDeductionInterval = 2; // Every 2 time ticks (30 minutes)
    [SerializeField] private int healthDeductionFromHunger = 2;
    [SerializeField] private int healthDeductionFromThirst = 3;

    [Header("UI References")]
    [SerializeField] HappinessBar healthBar;
    [SerializeField] HappinessBar happinessBar;
    [SerializeField] HappinessBar hungerBar;
    [SerializeField] HappinessBar thirstBar;

    [Header("Status")]
    public bool isDead;

    private DisableControls disableControls;
    private TimeAgent timeAgent;
    private int timeTickCounter = 0;

    private void Awake()
    {
        disableControls = GetComponent<DisableControls>();

        // Get or add TimeAgent component
        timeAgent = GetComponent<TimeAgent>();
        if (timeAgent == null)
        {
            timeAgent = gameObject.AddComponent<TimeAgent>();
        }
    }

    private void Start()
    {
        UpdateAllBars();

        // Subscribe to time events
        if (timeAgent != null)
        {
            timeAgent.onTimeTick += OnTimeTick;
        }
    }

    private void OnDestroy()
```

```
{
   // Unsubscribe from time events
   if (timeAgent != null)
   {
      timeAgent.onTimeTick -= OnTimeTick;
   }
}

private void OnTimeTick(DayTime dayTime)
{
   if (isDead) return;

   timeTickCounter++;

   // Deduct thirst more frequently than hunger
   if (timeTickCounter % thirstDeductionInterval == 0)
   {
      DeductThirst(thirstDeductionAmount);
   }

   if (timeTickCounter % hungerDeductionInterval == 0)
   {
      DeductHunger(hungerDeductionAmount);
   }

   // Check if hunger or thirst are empty and deduct health
   if (Hunger.IsEmpty())
   {
      DeductHealth(healthDeductionFromHunger);
   }

   if (Thirst.IsEmpty())
   {
      DeductHealth(healthDeductionFromThirst);
   }
}

private void Dead()
{
   if (!isDead)
   {
      isDead = true;
      if (disableControls != null)
      {
         disableControls.DisableControl();
      }
      Debug.Log("Character has died!");
   }
}
```

```
    private void CheckDeath()
    {
        if (Health.IsEmpty() || Happiness.IsEmpty())
        {
            Dead();
        }
    }

    private void UpdateAllBars()
    {
        UpdateHealthBar();
        UpdateHappinessBar();
        UpdateHungerBar();
        UpdateThirstBar();
    }

    private void UpdateHealthBar()
    {
        if (healthBar != null)
            healthBar.Set(Health.currVal, Health.maxVal);
    }

    private void UpdateHappinessBar()
    {
        if (happinessBar != null)
            happinessBar.Set(Happiness.currVal, Happiness.maxVal);
    }

    private void UpdateHungerBar()
    {
        if (hungerBar != null)
            hungerBar.Set(Hunger.currVal, Hunger.maxVal);
    }

    private void UpdateThirstBar()
    {
        if (thirstBar != null)
            thirstBar.Set(Thirst.currVal, Thirst.maxVal);
    }

    // Health Methods
    public void DeductHealth(int amount)
    {
        if (isDead) return;

        Health.Subtract(amount);
        UpdateHealthBar();
        CheckDeath();
    }

    public void AddHealth(int amount)
    {
        if (isDead) return;

        Health.Addition(amount);
        UpdateHealthBar();
    }

    public void FullHealth()
    {
        if (isDead) return;

        Health.SetToMax();
        UpdateHealthBar();
    }

    // Happiness Methods
    public void DeductHappiness(int amount)
    {
        if (isDead) return;

        Happiness.Subtract(amount);
        UpdateHappinessBar();
        CheckDeath();
    }

    public void AddHappiness(int amount)
    {
        if (isDead) return;

        Happiness.Addition(amount);
        UpdateHappinessBar();
    }

    public void FullHappiness()
    {
        if (isDead) return;

        Happiness.SetToMax();
        UpdateHappinessBar();
    }

    // Hunger Methods
    public void DeductHunger(int amount)
    {
        if (isDead) return;
        Hunger.Subtract(amount);
        UpdateHungerBar();
    }

    public void AddHunger(int amount)
    {
        if (isDead) return;

        Hunger.Addition(amount);
        UpdateHungerBar();
    }

    public void FullHunger()
    {
        if (isDead) return;

        Hunger.SetToMax();
        UpdateHungerBar();
    }

    // Thirst Methods
    public void DeductThirst(int amount)
    {
        if (isDead) return;

        Thirst.Subtract(amount);
        UpdateThirstBar();
    }

    public void AddThirst(int amount)
    {
        if (isDead) return;

        Thirst.Addition(amount);
        UpdateThirstBar();
    }

    public void FullThirst()
    {
        if (isDead) return;

        Thirst.SetToMax();
        UpdateThirstBar();
    }

    private void Update()
    {
        // You can add other passive updates here if needed
    }
```

```
}
```

CharacterController2D.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
public class CharacterController2D : MonoBehaviour
{
   Rigidbody2D rigidbody2d;
   [SerializeField] float speed = 2f;
   [SerializeField] float runSpeed = 5f;
   Vector2 motionVector;
   public Vector2 lastMotionVector;
   Animator animator;
   public bool moving;
   bool running;
   bool manualAnimationControl = false; // Flag to prevent Update from controlling
animations

   void Awake()
   {
      rigidbody2d = GetComponent<Rigidbody2D>();
      animator = GetComponent<Animator>();
   }

   private void Update()
   {
      if (Input.GetKeyDown(KeyCode.LeftShift))
      {
         running = true;
      }
      if (Input.GetKeyUp(KeyCode.LeftShift))
      {
         running = false;
      }

      float horizontal = Input.GetAxisRaw("Horizontal");
      float vertical = Input.GetAxisRaw("Vertical");

      motionVector = new Vector2(horizontal, vertical);

      // Only update animator if not under manual control
      if (!manualAnimationControl)
      {
         animator.SetFloat("horizontal", horizontal);
         animator.SetFloat("vertical", vertical);
```

```
            moving = horizontal != 0 || vertical != 0;
            animator.SetBool("moving", moving);

            if (horizontal != 0 || vertical != 0)
            {
                lastMotionVector = new Vector2(horizontal, vertical).normalized;
                animator.SetFloat("lastHorizontal", horizontal);
                animator.SetFloat("lastVertical", vertical);
            }
        }
    }

    void FixedUpdate()
    {
        // Only apply physics-based movement if not under manual control
        if (!manualAnimationControl)
        {
            Move();
        }
    }

    private void Move()
    {
        rigidbody2d.linearVelocity = motionVector * (running == true ? runSpeed : speed);
    }

    private void OnDisable()
    {
        rigidbody2d.linearVelocity = Vector2.zero;
    }

    public void WalkDown()
    {
        StartCoroutine(SmoothWalkDown());
    }

    private IEnumerator SmoothWalkDown()
    {
        // Take manual control of animations and physics
        manualAnimationControl = true;
        rigidbody2d.linearVelocity = Vector2.zero;

        Vector3 startPos = transform.position;
        Vector3 endPos = startPos + new Vector3(0, -1.5f, 0);
        float duration = 0.8f;

        // Set animation to show downward movement
        animator.SetFloat("lastHorizontal", 0);
        animator.SetFloat("lastVertical", -1);
        animator.SetBool("moving", true);

        // Update last motion vector for consistency
        lastMotionVector = Vector2.down;

        for (float t = 0; t < duration; t += Time.deltaTime)
        {
            transform.position = Vector3.Lerp(startPos, endPos, t / duration);
            yield return null;
        }

        // Ensure we reach the exact end position
        transform.position = endPos;

        // Stop the movement animation and return control to Update
        animator.SetBool("moving", false);
        manualAnimationControl = false;
    }
}
```

CharacterInteract.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CharacterInteract : MonoBehaviour
{
    CharacterController2D characterController;
    Rigidbody2D rgbd2d;
    [SerializeField] float offsetDistance = 1f;
    [SerializeField] float sizeOfInteractableArea = 1.2f;
    Character character;
    [SerializeReference] HighlightController highlightController;

    private void Awake()
    {
        characterController = GetComponent<CharacterController2D>();
        rgbd2d = GetComponent<Rigidbody2D>();
        character = GetComponent<Character>();
    }

    private void Update()
    {
        Check();
        if (Input.GetMouseButtonDown(1))
        {
            Interact();
        }
    }
    public void Check()
    {
        // Check for interactables around the character
        Interactable foundInteractable = FindNearestInteractable();

        if (foundInteractable != null)
        {
            highlightController.Highlight(foundInteractable.gameObject);
        }
        else
        {
            highlightController.Hide();
        }
    }

    private void Interact()
    {
        // Find and interact with the nearest interactable
        Interactable foundInteractable = FindNearestInteractable();

        if (foundInteractable != null)
        {
            foundInteractable.Interact(character);
        }
    }

    private Interactable FindNearestInteractable()
    {
        // Check both at character position and offset position
        Vector2 characterPosition = rgbd2d.position;
        Vector2 offsetPosition = rgbd2d.position + characterController.lastMotionVector *
offsetDistance;

        // First check at character's actual position
        Collider2D[] colliders = Physics2D.OverlapCircleAll(characterPosition,
sizeOfInteractableArea);
        Interactable nearestInteractable = null;
        float nearestDistance = float.MaxValue;

        foreach (Collider2D c in colliders)
        {
            Interactable interactable = c.GetComponent<Interactable>();
            if (interactable != null)
            {
                float distance = Vector2.Distance(characterPosition, c.transform.position);
                if (distance < nearestDistance)
                {
                    nearestDistance = distance;
                    nearestInteractable = interactable;
                }
            }
        }

        // If no interactable found at character position, check offset position
        if (nearestInteractable == null)
        {
            colliders = Physics2D.OverlapCircleAll(offsetPosition, sizeOfInteractableArea);
            foreach (Collider2D c in colliders)
            {
                Interactable interactable = c.GetComponent<Interactable>();
                if (interactable != null)
                {
                    float distance = Vector2.Distance(characterPosition, c.transform.position);
                    if (distance < nearestDistance)
                    {
                        nearestDistance = distance;
                        nearestInteractable = interactable;
                    }
                }
            }
        }

        return nearestInteractable;
    }
}
```

Currency.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Currency : MonoBehaviour
{
  [SerializeField] float amount;
  [SerializeField] TMPro.TextMeshProUGUI InventoryText;
  [SerializeField] TMPro.TextMeshProUGUI BillText;

  [Header("Achievement Tracking")]
  [SerializeField] private float totalEarnings; // Track total earnings separately
  [SerializeField] private AchievementManager achievementManager; // Reference to
achievement manager

  [Header("Testing Controls")]
  [SerializeField] private bool enableTestingMode = false; // Toggle this in inspector during
testing
  [SerializeField] private KeyCode addMoneyKey = KeyCode.M;
```

```
// Achievement constants
private const int FIRST_STEP_BILLIONAIRE_ID = 0;
private const float BILLIONAIRE_TARGET = 10000f;

private void Start()
{
   // Load saved total earnings first
   LoadTotalEarnings();

   // Find AchievementManager if not assigned
   if (achievementManager == null)
   {
      achievementManager = FindFirstObjectByType<AchievementManager>();
   }

   UpdateText();

   // Check if achievement should already be unlocked
   CheckBillionaireAchievement();
}

private void Update()
{
   // Only enable testing controls if testing mode is active
   if (enableTestingMode)
   {
      // Check for money cheat key
      if (Input.GetKeyDown(addMoneyKey))
      {
         Add(1000f);
      }
   }
}

private void UpdateText()
{
   InventoryText.text = amount.ToString("F2"); // Format to 2 decimal places
   BillText.text = amount.ToString("F2");
}

internal void Add(float moneyGain)
{
   if (moneyGain > 0) // Only count positive gains towards total earnings
   {
      amount += moneyGain;
      totalEarnings += moneyGain; // Add to cumulative earnings

      // Save total earnings
```

```
            SaveTotalEarnings();

            // Check achievement after earning money
            CheckBillionaireAchievement();

            string logPrefix = enableTestingMode ? "[TESTING] " : "";
        }

        UpdateText();
    }

    internal bool Check(float totalPrice)
    {
        return amount >= totalPrice;
    }

    internal void Decrease(float totalPrice)
    {
        amount -= totalPrice;
        if (amount < 0)
        {
            amount = 0;
        }
        UpdateText();

        // Note: We don't subtract from totalEarnings when spending money
        // because the achievement tracks total earned, not current balance
    }

    // Additional getter for current amount
    public float GetAmount()
    {
        return amount;
    }

    // Getter for total earnings
    public float GetTotalEarnings()
    {
        return totalEarnings;
    }

    // Method to set amount directly (useful for testing or loading save data)
    public void SetAmount(float newAmount)
    {
        amount = newAmount;
        if (amount < 0)
        {
            amount = 0;
        }
        UpdateText();
    }

    // Method to set total earnings directly (useful for loading save data)
    public void SetTotalEarnings(float earnings)
    {
        totalEarnings = earnings;
        if (totalEarnings < 0)
        {
            totalEarnings = 0;
        }

        SaveTotalEarnings();
        CheckBillionaireAchievement();
    }

    // Check and unlock the billionaire achievement
    private void CheckBillionaireAchievement()
    {
        if (achievementManager != null && totalEarnings >= BILLIONAIRE_TARGET)
        {
            // Only complete if not already completed (prevents duplicate popups)
            if
(!achievementManager.IsAchievementCompleted(FIRST_STEP_BILLIONAIRE_ID))
            {
                achievementManager.CompleteAchievement(FIRST_STEP_BILLIONAIRE_ID);
                string logPrefix = enableTestingMode ? "[TESTING] " : "";
            }
        }
    }

    // Save total earnings to PlayerPrefs
    private void SaveTotalEarnings()
    {
        PlayerPrefs.SetFloat("TotalEarnings", totalEarnings);
        PlayerPrefs.Save();
    }

    // Load total earnings from PlayerPrefs
    private void LoadTotalEarnings()
    {
        totalEarnings = PlayerPrefs.GetFloat("TotalEarnings", 0f);
    }

    // UPDATED: Reset testing progress AND the achievement (but keep it unlocked)
    private void ResetTestingProgress()
    {
        if (!enableTestingMode)
        {
            return;
        }

        // Reset the earnings
        totalEarnings = 0f;
        SaveTotalEarnings();

        // Reset the achievement but keep it unlocked (so it can be completed again for testing)
        if (achievementManager != null)
        {
            // Find the achievement and set it to unlocked but not completed
            var achievement = achievementManager.achievements.Find(a => a.id ==
FIRST_STEP_BILLIONAIRE_ID);
            if (achievement != null)
            {
                achievement.isUnlocked = true;
                achievement.isCompleted = false;

                // Save the new state
                achievementManager.SaveAchievements();

                // Refresh the display
                achievementManager.ForceRefreshAllDisplays();

            }
        }
        else
        {
            Debug.LogWarning("[TESTING] AchievementManager not found - only reset
earnings");
        }
    }

    // Context menu methods for testing in editor
    [ContextMenu("Add Test Earnings (1000)")]
    public void AddTestEarnings()
    {
        Add(1000f);
    }

    [ContextMenu("Check Achievement Progress")]
    public void CheckAchievementProgress()
    {
        float progress = (totalEarnings / BILLIONAIRE_TARGET) * 100f;
        bool isCompleted = achievementManager != null ?
achievementManager.IsAchievementCompleted(FIRST_STEP_BILLIONAIRE_ID) : false;
        Debug.Log($"Billionaire Achievement Progress: {totalEarnings:F2} /
{BILLIONAIRE_TARGET:F2} ({progress:F1}%) - Completed: {isCompleted}");
    }
    [ContextMenu("Reset Total Earnings (Testing Only)")]
    public void ResetTotalEarnings()
    {
        if (enableTestingMode || Application.isEditor)
        {
            ResetTestingProgress();
        }
        else
        {
            Debug.LogWarning("Cannot reset earnings - testing mode is disabled and not in
editor");
        }
    }

    [ContextMenu("Force Reset Achievement (Keep Unlocked)")]
    public void ForceResetAchievementKeepUnlocked()
    {
        if (achievementManager != null)
        {
            var achievement = achievementManager.achievements.Find(a => a.id ==
FIRST_STEP_BILLIONAIRE_ID);
            if (achievement != null)
            {
                achievement.isUnlocked = true;
                achievement.isCompleted = false;

                // Save the new state
                achievementManager.SaveAchievements();

                // Refresh the display
                achievementManager.ForceRefreshAllDisplays();

                Debug.Log("Reset billionaire achievement to unlocked but not completed");
            }
        }
        else
        {
            Debug.LogWarning("AchievementManager not found");
        }
    }

    [ContextMenu("Toggle Testing Mode")]
    public void ToggleTestingMode()
    {
        enableTestingMode = !enableTestingMode;
        Debug.Log($"Testing mode {(enableTestingMode ? "ENABLED" : "DISABLED")}");
    }
}
```

DayTime.cs:
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Rendering.Universal;
using UnityEngine.UI;

public enum DayOfWeek
{
   Sunday,
   Monday,
   Tuesday,
   Wednesday,
   Thursday,
   Friday,
   Saturday
}

public class DayTime : MonoBehaviour
{
   const float secondsInDay = 86400f;
   const float phaseLength = 900f; // 15 minutes chunk of time
   const float phasesInDay = 96f; //seconsInDay divided by phaseLength

   [SerializeField] Color nightLightColor;
   [SerializeField] AnimationCurve nightTimeCurve;
   [SerializeField] Color dayLightColor = Color.white;

   float time;
   [SerializeField] float timeScale = 60f;
   [SerializeField] float startAtTime = 21600f; // in seconds
   [SerializeField] float morningTime = 21600f;

   DayOfWeek dayOfWeek;

   [SerializeField] TMPro.TextMeshProUGUI text;
   [SerializeField] Text dayOfTheWeekText;
   [SerializeField] TMPro.TextMeshProUGUI dayCountText;
   [SerializeField] Light2D globalLight;

   [Header("Time-based Sprite")]
   [SerializeField] Image timeSprite;
   [SerializeField] Sprite sprite6AM;     // Early Morning
   [SerializeField] Sprite sprite12PM;    // Noon
   [SerializeField] Sprite sprite3PM;     // Afternoon
   [SerializeField] Sprite sprite5PM;     // Late Afternoon
   [SerializeField] Sprite sprite7PM;     // Evening
   [SerializeField] Sprite sprite12AM;    // Midnight
```

```
    public int days;
    List<TimeAgent> agents;

    // Singleton pattern
    public static DayTime Instance { get; private set; }

    private void Awake()
    {
        // Singleton setup
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
            return;
        }

        agents = new List<TimeAgent>();
    }

    private void Start()
    {
        time = startAtTime;
        UpdateDayText();
        UpdateDayCountText();
        UpdateTimeSprite();
    }

    private void UpdateTimeSprite()
    {
        if (timeSprite == null) return;

        float currentHour = Hours;

        // Determine which sprite to show based on current time
        if (currentHour >= 0f && currentHour < 6f)
        {
            // Midnight to 6AM - show midnight sprite
            timeSprite.sprite = sprite12AM;
        }
        else if (currentHour >= 6f && currentHour < 12f)
        {
            // 6AM to 12PM - show morning sprite
            timeSprite.sprite = sprite6AM;
        }
        else if (currentHour >= 12f && currentHour < 15f)
        {
            // 12PM to 3PM - show noon sprite
            timeSprite.sprite = sprite12PM;
        }
        else if (currentHour >= 15f && currentHour < 17f)
        {
            // 3PM to 5PM - show afternoon sprite
            timeSprite.sprite = sprite3PM;
        }
        else if (currentHour >= 17f && currentHour < 19f)
        {
            // 5PM to 7PM - show late afternoon sprite
            timeSprite.sprite = sprite5PM;
        }
        else if (currentHour >= 19f && currentHour < 24f)
        {
            // 7PM to Midnight - show evening sprite
            timeSprite.sprite = sprite7PM;
        }
    }

    private void UpdateDayCountText()
    {
        if (dayCountText != null)
        {
            dayCountText.text = (days + 1).ToString("00");
        }
    }

    public void Subscribe(TimeAgent timeAgent)
    {
        agents.Add(timeAgent);
    }

    public void Unsubsribe(TimeAgent timeAgent)
    {
        agents.Remove(timeAgent);
    }

    public float Hours
    {
        get
        {
            return time / 3600f;
        }
    }

    public float Minutes
```

```
{
   get
   {
      return time % 3600f / 60f;
   }
}

private void Update()
{
   time += Time.deltaTime * timeScale;
   TimeValueCalculation();
   DayLight();
   UpdateTimeSprite();

   if (time > secondsInDay)
   {
      NextDay();
   }

   TimeAgents();

   if (Input.GetKeyDown(KeyCode.R))
   {
      SkipTime(hours: 4);
   }
}

int oldPhase = -1;

private void TimeAgents()
{
   if(oldPhase == -1)
   {
      oldPhase = CalculatePhase();
   }

   int currentPhase = CalculatePhase();

   while(oldPhase < currentPhase)
   {
      oldPhase += 1;
      for (int i = 0; i < agents.Count; i++)
      {
         agents[i].Invoke(this);
      }
   }
}

private int CalculatePhase()
```

```
    {
        return (int)(time / phaseLength) + (int)(days * phasesInDay);
    }

    private void TimeValueCalculation()
    {
        int hh = (int)Hours;
        int mm = (int)Minutes;
        text.text = hh.ToString("00") + ":" + mm.ToString("00");
    }

    private void DayLight()
    {
        float v = nightTimeCurve.Evaluate(Hours);
        Color c = Color.Lerp(dayLightColor, nightLightColor, v);
        globalLight.color = c;
    }

    private void NextDay()
    {
        time -= secondsInDay;
        days += 1;

        int dayNum = (int)dayOfWeek;
        dayNum += 1;
        if (dayNum >= 7)
        {
            dayNum = 0;
        }
        dayOfWeek = (DayOfWeek)dayNum;

        UpdateDayText();
        UpdateDayCountText();
    }

    private void UpdateDayText()
    {
        string chineseDayText = "";
        switch (dayOfWeek)
        {
            case DayOfWeek.Sunday:
                chineseDayText = "日";
                break;
            case DayOfWeek.Monday:
                chineseDayText = "一";
                break;
            case DayOfWeek.Tuesday:
                chineseDayText = "二";
                break;
            case DayOfWeek.Wednesday:
                chineseDayText = "三";
                break;
            case DayOfWeek.Thursday:
                chineseDayText = "四";
                break;
            case DayOfWeek.Friday:
                chineseDayText = "五";
                break;
            case DayOfWeek.Saturday:
                chineseDayText = "六";
                break;
        }
        dayOfTheWeekText.text = chineseDayText;
    }

    public void SkipTime(float seconds = 0, float minute = 0, float hours = 0)
    {
        float timeToSkip = seconds;
        timeToSkip += minute * 60f;
        timeToSkip += hours * 3600f;
        time += timeToSkip;
    }

    public void SkipToMorning()
    {
        float secondsToSkip = 0f;
        if(time > morningTime)
        {
            secondsToSkip += secondsInDay - time + morningTime;
        }
        else
        {
            secondsToSkip += morningTime - time;
        }
        SkipTime(secondsToSkip);
    }

    // Public methods for accessing time data from other scenes
    public string GetTimeString()
    {
        int hh = (int)Hours;
        int mm = (int)Minutes;
        return hh.ToString("00") + ":" + mm.ToString("00");
    }

    public DayOfWeek GetDayOfWeek()
    {
        return dayOfWeek;
    }

    /// Gets the current time scale multiplier
    public float TimeScale
    {
        get { return timeScale; }
    }
    /// <param name="scale">The new time scale (0 = paused, 1 = normal speed, >1 =
faster)</param>
    public void SetTimeScale(float scale)
    {
        timeScale = scale;
    }
    /// </summary>
    public void PauseTime()
    {
        timeScale = 0f;
    }

    /// <param name="scale">Optional: specific scale to resume with, defaults to 60f</param>
    public void ResumeTime(float scale = 100f)
    {
        timeScale = scale;
    }
}
```

GameManager.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class GameManager : MonoBehaviour
{
    public static GameManager instance;
    private void Awake()
    {
        instance = this;
    }
    public GameObject player;
    public ItemContainer inventoryContainer;
    public ItemDragAndDropController dragAndDropController;
    public DialogueSystem dialogueSystem;
    public DayTime timeController;
    public ScreenTint screenTint;
    public PlaceableObjectsReferenceManager placeableObjects;
    public ItemStockPanel stockStorePanel;
    public OptionDialogueSystem optionDialogueSystem;
    public DialogueActionHandler dialogueActionHandler;
    public AchievementManager achievementManager;
}
```

POSTER

FINANCE JOURNEY IN UNIVERSITY

Prepared by Dickson Shee Wei Hau

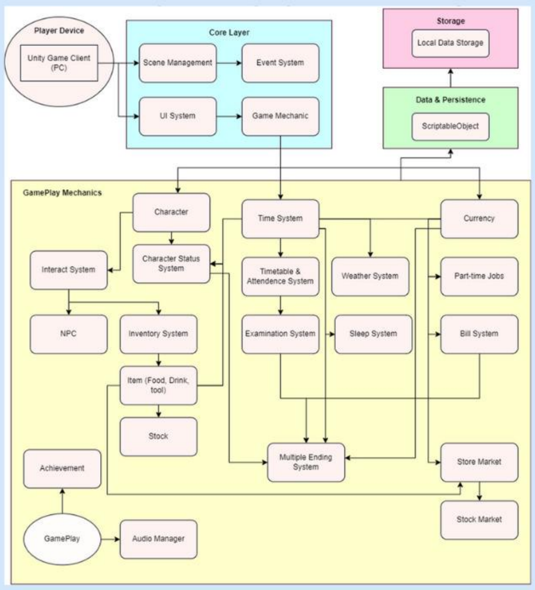
INTRODUCTION

This project presents Finance Journey in University, a 2D top-down pixel art educational game designed to improve financial literacy among university and college students. Financial literacy is not only knowledge of money management but also the confidence and ability to apply that knowledge effectively in real-life decision-making. Studies have shown that young adults aged 18 to 24 especially university students often demonstrate the lowest levels of financial literacy by leaving them vulnerable to overspending, debt and financial stress. Rising education costs and limited income opportunities further increase the risk of poor money management, making innovative and accessible solutions urgently needed.

OBJECTIVES

- **Enhance Financial Literacy** - Provide 2D educational game that helps university and college students learn to manage money effectively in their daily lives.
- **Teach Core Financial Concepts** - Integrate core financial concepts into decision-making scenarios to highlight to impact of short and long-term financial choices.
- **Motivate Financial Goal** - Encourage students connect financial literacy with personal aspirations to supports dreams

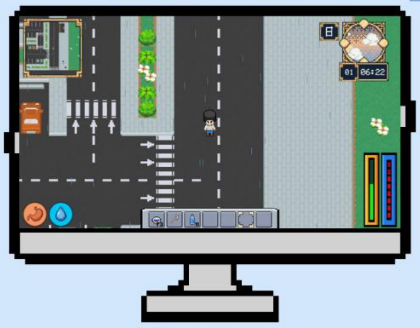
METHODOLOGY



- **Player Device** - Runs the Unity game client on PC, providing the platform for players to interact with the game.
- **Core Layer** - Manages essential game logic including scenes, events, UI, and mechanics.
- **Gameplay Mechanics** - Implements player activities such as character management, inventory, time system, exams, markets, and multiple endings.
- **Data & Persistence** - Handles data storage and retrieval using ScriptableObjects for efficient game state management.
- **Storage** - Maintains local data storage for saving player progress and game information.

SYSTEM EVALUATION AND DISCUSSION

GAME SCENE



- **Concept Feasibility** - project successfully demonstrates that a 2D simulation game can integrate financial literacy into an engaging and interactive learning environment
- **Core Gameplay Systems** - Fundamental mechanics such as movement, interaction, time cycles, inventory, wallet management, and market systems were effectively implemented, proving the technical feasibility of the design
- **Educational Value** - Financial concepts like budgeting, saving, and investing are embedded into realistic scenarios, while health and mental health systems encourage players to balance different aspects of life
- **Player Engagement** - Replayability is enhanced through a multiple-ending system, dynamic outcomes, and random opportunities that motivate players to explore diverse financial strategies

CONCLUSION

Overall, Finance Journey in University demonstrates the effectiveness of using a 2D simulation game as an innovative educational tool. The system achieved its objectives by integrating financial literacy concepts into realistic gameplay that ensuring both engagement and learning. While some limitations remain in content depth and game mechanics, the project has established a solid foundation that can be expanded in future development to further enrich player experience and educational value.