# DEVELOPMENT OF A TOWER-DEFENSE GAME WITH PROCEDURALLY GENERATED STAGES

A Thesis

Presented to the Faculty of the

Computer Studies Department

College of Science

Technological University of the Philippines

Ayala Boulevard, Manila

**In Partial Fulfillment of the**

**Requirements for the Degree**

**Bachelor of Science in Computer Science**

**By:**

Rosete, Keith C.

Eboña, Dan Eiaiel F.

Manibale, Cyrene Rose C.

Lagas, Maurice H.

**BSCS-4B-M**

**June 2024**

| TECHNOLOGICAL UNIVERSITY OF THE PHILIPPINES<br>Ayala Blvd., Ermita, Manila, 1000, Philippines \| Tel No. +632-5301-3001 local 608<br>Fax No. +632-8521-4063 \| Email: cos@tup.edu.ph \| Website: www.tup.edu.ph<br>THESIS APPROVAL SHEET FOR THE UNDERGRADUATE PROGRAMS OF THE COS | Index No | TUPM-F-COS-16-TAU |
|---|---|---|
| | Revision No | 00 |
| | Date | 07012022 |
| | Page | 1 / 1 |

This thesis hereto entitled:

**DEVELOPMENT OF A TOWER-DEFENSE GAME WITH PROCEDURALLY GENERATED STAGES**

prepared and submitted by **DAN ELAIEL F. EBOÑA, CYRENE ROSE C. MANIBALE, MAURICE H. LAGAS, KEITH C. ROSETE** in partial fulfillment of the requirements for the degree **BACHELOR OF SCIENCE IN COMPUTER SCIENCE** has been examined and is recommended for approval and acceptance.

**PROF. PEREGRINO B. AMADOR JR.**
Adviser

Approved by the Committee on Oral Examination with a grade of **PASSED** on JUNE 5, 2024.

**PROF. MAY GARCIA**
Member

**PROF. EDWARD N. CRUZ**
Member

**PROF. DOLORES L. MONTESINES**
Department Head/Chair

Accepted in partial fulfillment of the requirements for the degree **BACHELOR OF SCIENCE IN COMPUTER SCIENCE**.

Date: June 14, 2024

**DR. JOSHUA T. SORIANO**
Acting Dean

| Transaction ID | TUPM-COS-TAU-ELS-07012022 -0258PM |
|---|---|
| Signature | |

Transaction ID Legend: TUPX AAA (Office Code)-BBB (Type of Transaction)-CCC (Initial of employee)-MMDDDYYYY (month day year)-HHMMAM/PM (hourminutesAM/PM)

**Acknowledgement**

The researchers would like to express their sincere gratitude to everyone who have supported them throughout the process of completing this study. This work would not have been possible without their guidance, encouragement, and assistance.

First and foremost, the researchers would like to thank our thesis advisor, Professor Peragrino B. Amador Jr., for his unwavering support, invaluable guidance, and insightful feedback throughout this research. His expertise and encouragement have been instrumental in shaping this work.

We are also profoundly grateful to the panel members from the College of Science faculty, Professor May M. Garcia and Professor Edward N. Cruz, for their thoughtful comments, constructive criticism, and dedication to improving the quality of our research. Professor Edward N. Cruz is also the one who handles our thesis subject. Their guidance and understanding during the semester allowed us to be informed of what to do and not to do, which led us to finish our tasks.

We also want to express our gratitude and appreciation to our friends and family who have provided patience, and understanding. Believing in our ability to persevere towards the completion of this study is pivotal for the success of the whole research team.

# ABSTRACT

Tower Defense is a popular game genre where players need to defend against a wave of slime monsters. This study aims to generate maps for the Tower Defense game procedurally. The game is designed to use the Perlin Noise algorithm to place props on the map grid. The Ant Colony Algorithm is also used by the enemies in the Tower Defense game to add randomness to their behavior. The game was developed using the Unity Game Engine and the C# programming language. The functionality and maintainability of the game will be tested. Two cycles of tests will be executed. The functionality and maintainability tests yielded a 100% success rate on both cycles of testing. A survey was also conducted to help evaluate the game's functionality and maintainability. The respondents consist of 25 students and 6 IT professionals using purposive sampling. Based on the gathered data, the assessment result has an overall mean of 3.13 with a qualitative interpretation of Very Acceptable.

Keywords: tower defense, procedural generation, Perlin noise, ant colony, unity engine

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF APPENDICES

# Chapter 1

# THE PROBLEM AND ITS SETTINGS

## Background of the Study

The tower defense genre in video games has been popular in the gaming industry. Establishing itself as an option for both casual and non-casual gamers in combination of strategy and decision-making in real time. The core gameplay mechanic of a tower defense game is to protect the base or specified area from waves of enemies that progressively become more challenging in each round. (Masterclass, 2021)

Tower Defense (TD) is a sub-genre of Real-Time Strategy (RTS) games. Thus, the early days of the TD genre can be traced back between the years 2000 and 2010 from RTS games with some popular ones being created as custom maps in “Warcraft III” and the rise of Flash games such as “Flash Element TD” and “Desktop Defense.” With more popular standalone tower defense games such as “Plants vs. Zombies” and “Bloons Tower Defense” shown up. As more games in the genre increase along with the genre’s rapid growth and popularity, the challenge of creating more engaging and dynamic gameplay remains apparent.

Traditional tower defense games that rely on fixed paths and static basic enemy AI struggles to compete as it makes gameplay predictable and offers less replayability. Thus, developers started branching out by combining gameplay elements from other games such as First-Person Shooter (FPS) games and making enemy AI more complex such as attacking towers placed by players. There has also been an increase in allowing more interactions between the player and the game such as having the ability to place traps or obstructions to prevent enemies from getting through. Which effectively introduces more

things to learn in the gameplay as well as providing more replayability to the players. (Sticky 2006)

The purpose of this study is to explore methods of developing a tower defense game by implementing randomized path generation and adding complexity to the AI behavior of enemy Non-Playable Characters (NPCs). By addressing these ideas, the study aims to create a tower defense game experience that is different every time a game is started thus having more replayability and less predictability.

**Objectives of the Study**

The General Objective of this study is to develop a Tower Defense Game with procedurally generated stage generation with dynamic enemy AI pathfinding. Specifically, it aims to:

1. Design the Game Application with the following features:
    a. Implement a Procedural Content Generation system incorporating:
        i. Grid Generation
        ii. Randomized Placement of In-game Objects using Perlin Noise.
    b. Implement Ant Colony Optimization on Enemy Monster AI to diversify gameplay challenges
    c. Design Health Points and Currency Systems to introduce resource management mechanics.
    d. Design various types of monsters with unique characteristics to promote diverse strategies.

e. Design different tower types to encourage strategic choices and adaptability in gameplay.

2. Test and improve the game application based on Functional Suitability and Maintainability.
3. Determine the level of accessibility of the developed system using ISO/IEC 25010 Software Quality Metrics.

**Scope and Limitations of the Study**

The scope of this study is to focus on the development of a tower defense game with procedurally generated stages. The study aims to identify and create the necessary features of a fun and engaging Tower-Defense and to develop a procedurally generated map with waves of enemies.

The features to be used in this study are as follows:

**Main Menu -** comprises various graphical visuals such as a static background image, the title of the game, and several buttons for basic features of the game application. The “Play” button will prompt the game to open a panel containing the selectable game modes that the player can choose before a new map is generated. The “Options” button allows the user to change options such as the sound and music volume as well as the screen resolution of the game. There is also a gallery button which the user can navigate to read information about the playable towers and the slimes present in the game. And then the “Exit” button that closes the game.

**Health Points (HP) System** - in every playthrough, the player is only given a limited amount of HP. Whenever enemies manage to reach the endpoint, the player’s

remaining HP will go down and if it goes to 0 then its game over. The player will be sent to the game over screen. If the player has at least 1 remaining HP when the final wave ends, then the player will be shown a victory screen. Enemies will also have various HP counts depending on its type, if its HP goes down to zero then it is defeated.

**Gold (G) System -** this serves as the game's currency to be used for building and upgrading towers. These can be obtained by defeating enemies that spawn every wave and whenever a new wave starts.

The tower defenses game will be created using the Unity game engine and programming with C# through the use of Visual Studio Code. The game will only be developed on desktop computers and laptops. It will also be just an offline single player game that generates a new map whenever a new game is started through procedural generation.

This study will not include a story-campaign and stage selection features as the main focus is to generate procedural content for its main gameplay. Multiplayer mode will not be available as it would require the base game to also be made in consideration of playing it in multiplayer. In implementing the enemy monster AI, usage of algorithms that only calculate the shortest path such as A* algorithm will be avoided but may be used for other game mechanics such as ensuring that there are always valid paths between the spawn point and the target objective of the enemies. While the enemy AI may randomly choose the shortest path, it will still have the option to randomly select a less optimal path towards the target endpoint.

This study is limited to the development of a Tower Defense game with procedurally generated stages. The game will not focus on the shortest path possible. The A.I. may

randomly select the shortest path possible but the A.I. is allowed and preferred to take other paths. The AI used for the randomized generation will only focus on creating the playable areas of the game. All the other functions of the Tower Defense are created by the Researcher. The Programming Language used in the development of the study is C# utilized by Unity Game Engine. These limitations should be considered when interpreting the findings of this study and when applying the results to specific situations.

**Significance of the Study**

The Significance of this study is to research Human-A.I. Interactions. One of the advantages of the study is to note the reactions of an A.I. when all the tools available are either working or not working against the player. This can also show how A.I. plans and builds things with Procedural Generation. These topics can help further the research of A.I.

**User**. By Randomizing a few elements of gameplay such as Map Design as well as Track Pathing, the game offers more unpredictable encounters by reducing the number of enemies taking the shortest path possible as well as creating inconvenient spots of Tower placement areas. This offers more replayability in terms of placement planning as well as reducing the possibility of "META" building techniques such as only placing 1 type of tower as well as a specific but effective tower placement for all entirety of the gameplay.

**Researchers**. The Development of a Tower-Defense Game with Procedurally Generated Stage provides an opportunity for the Researchers to contribute to the improvement of games especially for Procedurally Generated Games. Through the design, development,

and implementation of the system, the researcher can gain valuable insights into the challenges faced by developers during project development. The study offers an opportunity to apply theoretical concepts, problem solving skills, and technical skills to overcome challenges to solve real world problems, causing the enhancement of the Researcher's skill and Professional Development. Additionally, the results of the study can serve as a basis for future research in the field of Procedurally Generated Games especially through the means of Tower Defense.

**Game Developers.** The development of this thesis project offers an example or proof of concept for other aspiring game developers who have similar ideas regarding the use of procedural content generation in their tower defense game. While it's not just limited to one genre of games, the application of certain algorithms and functions used in this project may give valuable ideas for fellow game developers. It can also provide potential solutions to coding problems that other developers may have encountered that they couldn't find elsewhere. This allows for additional materials to reference through the project when it comes to procedural content generation in video games.

**Students.** For other students that are aiming to go into game development, this thesis project offers a new potential reference material to be used. While it may vary depending on course requirements and thesis advisers, this project also provides students who are hesitant in choosing a thesis topic the idea that they can select game development as a potential topic for their thesis project. Computer Science or Information Technology students can also obtain valuable insight on what algorithms and artificial intelligence could be utilized for their proposed games as their thesis topic.

# Chapter 2

# CONCEPTUAL FRAMEWORK

The review of relevant studies and literature that support the study's background is covered in this chapter. Included are the study's conceptual framework and the operational definition of terms.

## Review of Related Literature

### *Computational Intelligence and Tower Defense Games*

Tower Defense (TD) games, as defined by Avery, et al., (2011), stands out as strategic experiences where players must strategically allocate resources and strategically place towers to thwart waves of adversaries. These games, known for their simplicity in gameplay coupled with challenging tactical elements, have garnered a significant following among casual gamers. The inherent simplicity and popularity of TD games, make them an ideal testing ground for Computational Intelligence (CI) research. The straightforward nature of TD games not only ensures a pleasurable gaming experience but also facilitates in-depth CI exploration, given the ease of calculation and graphics utilization.

TD games, with their tactical depth and varied gameplay mechanics, offer a rich landscape for AI research. The complexity introduced by multiple game types, diverse resource allocation mechanisms, and the option for turn-based or real-time gameplay opens up multiple avenues for CI experimentation. Moreover, the inclusion of online play features in some TD games not only broadens the scope of testing but also enhances user engagement. This makes TD games a logical and valuable medium for CI research,

presenting a unique blend of simplicity and strategic depth that appeals to both researchers and gamers alike.

### *Artificial Intelligence in Games and Its Application*

Artificial Intelligence (AI) in gaming serves as the emulation of human intelligence by machines, acting as the cognitive powerhouse behind various game functions. In the context of gaming, AI takes on the role of the 'brains' of the game, influencing aspects such as creature behavior and weapon mechanics. Tank (2023) delves into the multifaceted role of AI in gaming architecture, encompassing robotics, natural language processing, speech and visual recognition, planning, learning, problem-solving, and knowledge representation.

According to Tank's (2023) paper, AI's significance extends beyond mere functionality, playing a pivotal role in enhancing realism across graphics, physics, non-player character behavior, and overall game systems. The incorporation of AI into these facets contributes to a heightened gaming experience, amplifying gameplay dynamics and transforming the gaming landscape. This synthesis encapsulates the profound influence of AI in the gaming industry, showcasing its diverse contributions that go beyond mere functionality to shape and elevate the entire gaming scene.

### *Procedural Generation of Tower Defense Levels*

Tower defense games stand out as strategic digital experiences, requiring players to skillfully protect bases against enemy attacks following meticulously planned routes. This genre's gameplay hinges on the delicate balance between offense and defense, demanding precise control over the placement of defensive structures along predetermined pathways.

In a study by Öhman (2020), a pivotal advancement in tower defense game development was introduced through the integration of a straightforward algorithm into the path generation system. This algorithm serves with dual purpose: determining the quantity of enemies and strategically situating defense towers. The net result is a tower defense game characterized by simplicity, yet its level generation system is thorough, albeit exhibiting some inherent unevenness. Despite the apparent imbalance in the generated levels, it is crucial to underscore the intentional design of the system to be adaptable. The flexibility of this system allows for dynamic adjustments, emphasizing the pursuit of balance in gameplay. The decision to deviate from the original procedural plan stemmed from pragmatic considerations related to time constraints. This modification, however, should not be misconstrued as a deficiency in the initial proposal's procedures; rather, it underscores the imperative for a more time-efficient strategy within the specific scope of the project.

In essence, the paper underscores the dynamic nature of tower defense game development, where strategic adaptability and efficiency are paramount considerations in achieving a balanced and engaging gaming experience.

***Evaluating the Impact of Procedurally Generated Content on Game Immersion***

Connor, et al., (2017) delved into the evaluation of Procedurally Generated Content (PGC) on game immersion, emphasizing its crucial role in player engagement. The creation of game content, especially in the context of increasingly complex games, presents challenges in terms of both development and cost. Manual content creation is not only expensive but also lacks scalability, prompting the exploration of alternatives such as Procedural Content Generation (PCG).

PCG, leveraging algorithmic content creation with minimal human intervention, offers potential cost-saving advantages. However, its implementation is not without challenges, including computational overhead and the need to align with technical and cultural values in content generation. The study, as outlined in the abstract, seeks to explore the impact of PCG by conducting a comparative analysis between human-designed and procedurally generated game levels. This investigation aims to shed light on how the utilization of PCG influences the overall gaming experience.

This review critically examines the potential benefits and challenges associated with PCG, setting the stage for the in-depth analysis carried out in the study. By navigating through the intricacies of game content creation, the research aims to contribute valuable insights into the influence of procedural generation on gameplay immersion.

### *Game Design Research: An Overview*

Holopainen and Lankoski (2017) offered a nuanced exploration of the evolution of game design research, emphasizing its historical trajectory and fundamental role as a problem-solving discipline within the gaming domain. Distinguishing between game design research and the broader field of game research, which encompasses studies on games, play, or players, the document traces the less-explored history of game design studies. Pivotal moments, such as Chris Crawford's influential work in the 1980s and early publications focusing on game design, serve as key milestones in the field's development.

The overview provided by Holopainen and Lankoski (2017) underscores the multidimensional nature of design research, categorizing it into three distinct dimensions: epistemology, praxeology, and phenomenology. These dimensions delve into the intricate realms of designer knowledge, practices, and resulting artifacts, contributing to a holistic understanding of the discipline. Furthermore, the document delineates three specific categories within design research: research on design, research for design, and research through design. Each category serves a unique purpose, aiming to comprehend, develop, and produce knowledge relevant to the intricate landscape of game design.

In summary, the authors' comprehensive narrative serves as a reflective overview of game design research, offering valuable insights into its historical evolution and multifaceted nature. By delving into the complexities of design research, their work provides a foundational resource for scholars and practitioners, enriching our understanding of the dynamic and evolving field of game design.

***Performance Evaluation of Pathfinding Algorithms***

Pathfinding algorithms play a critical role in Artificial Intelligence, serving as integral components for determining optimal routes in applications ranging from gaming and robotics to navigation systems. This study highlights the various factors influencing the performance of these algorithms, including considerations of problem size, obstacles, and heuristic structures. An in-depth exploration into the empirical evaluation of newly proposed algorithms is conducted, emphasizing the significance of robust experimental design and thorough analysis. (Sidhu, 2019).

Through a comprehensive survey of existing research, the abstract identifies common weaknesses in methodologies employed in pathfinding algorithm studies. Additionally, the study provides practical solutions by presenting illustrative problems designed to rectify these identified shortcomings. The research goes further to illuminate potential misleading elements such as spurious effects, control conditions, and sampling biases. To address these concerns, the study offers insightful case studies, aiming to mitigate these pitfalls commonly encountered in pathfinding studies (Sidhu, 2019).

In summary, this contributes to our understanding of the critical nature of pathfinding algorithms, providing valuable insights into their performance evaluation and addressing key methodological challenges. Sidhu's work serves as a valuable resource for researchers and practitioners in the field of Artificial Intelligence, offering guidance for the effective development and assessment of pathfinding algorithms in various applications.

***Perlin Noise***

Perlin noise is a set of controlled randomized numbers that has been smoothed to be used in procedural generation. The Perlin noise starts as a set of numbers representing a map of variables expressing maximum and minimum values. (Ebert et al., 2003). These sets of numbers are similar to a white noise system however these sets of numbers are controlled so that the minimum variables are spread out to create a map of minimum variables surrounded by maximum variables. Finally, the values between the minimum variables are smoothened, resulting in a Perlin Noise Map.

The Perlin Noise Map could be used in many sorts of media from game development to art, a popular concept that is regularly used in programming is a Height Map that models as a three-dimensional map with its maximum variable representing the highest point allowed and the minimum variable representing the lowest point within the model. A well-known media that is known to use this concept is a game called Minecraft. The zeroes could represent the minimum block allowed while the values that remained one represents the maximum block allowed, while the other values smoothened out can be expressed as the blocks between smoothening out the ones from the zeroes. (Ebert et al., 2003)

### *Pixel Art*

According to Technopedia (2015), Pixel art is a type of digital art in which graphics editing software is used to create and modify images at the pixel level. Individual pixels act as the building blocks of the image in pixel art, which has its own distinct visual style. The outcome is a visual aesthetic that is strikingly reminiscent of cross-stitch, mosaic art, and other embroidery techniques.

According to Tokio School (2023), with its nostalgic connections to video games from the 1980s and 1990s, Pixel Art is a style that is firmly entrenched in the history of gaming. The undeniable charm of pixelated digital artwork acts as a persistent reminder of the game heritage, even in the face of technology improvements and creative design methods. In the modern era, this vintage look is becoming more and more important. The enduring passion for old-school video games isn't just a passing fad—rather, it seems to be developing into a complementary and substitute kind of online pleasure. The vintage appeal of Pixel Art draws in both novice and expert gamers, which fuels the growth of the gaming industry and increases the need for qualified workers.

### *Minecraft*

Minecraft uses multiple Perlin Noise in order to create the multiple Worlds of Minecraft. Minecraft's Map Generator first uses the Biome map. This dictates the different biomes that will be spread throughout the area. As of 2022, there are 60 different biomes that are available on the Overworld, some of which are the Plains, Desert, Mountains and Ocean Biome. Next Minecraft uses Perlin Noise to create a Height Map in order to facilitate

the creation of the basic outline of an area. If an area outside the Mountain Biome is populated by White Noise, then the Height Map designates it as the Regional Hills whilst Any areas that are lower than the minimum height will be turned into water. This ranges from underground reservoirs to regional ponds. The next step for the Height map is to designate all Surface blocks into Blocks appropriate for their biome. Grass Blocks for the Plains, Sand Blocks for the Desert, as well as and Snow for the Tundra Biome etc. Then the third and final phase gives the final touch by inhabiting the remaining features of the game from the mobs, villages, strongholds and more. (Zucconi, 2023)

### *Sid Meier's Civilization*

Sid Meier's Civilization Uses Perlin Noise in order to create a tile-based 2-dimensional Map for its game. The Perlin Noise starts by giving a value to a threshold and setting a rule where everything equal or above it is land and everything lower than it is water ranging from shallow waters to deep ocean floor. When a value reaches a certain value, it will change terrain to mountains and hills. And if a certain value is too low it will block access for certain tiles around it, preventing units from traversing. Sid Meier's Civilization also uses Perlin Noise for its resource, giving special resources to players that acquire it in white noise and preventing another white noise while it is still within the certain range from the white noise. (Coding Quest, 2023)

### *Ant Colony Optimization Algorithm*

Ant Colony Optimization Algorithm is a concept that mimics a real ant colony's behavior for finding its needed resource, usually food, and bringing it back to their colony

as fast as possible. This is achieved by sending multiple ants taking different routes towards the same destination each of which leaving trails of odorless pheromones detectable by other ants going to and from its desired destination. The shorter path taken has stronger pheromones due to ants leaving behind multiple pheromones caused by multiple trips made because of a shorter trip. Eventually, the shortest path is the only path taken because of the largest number of pheromones caused by the bias of multiple ants taking multiple trips (Dorigo & Stützle, 2004).

A computer tries to simulate this by sending signals towards the multiple receivers in multiple routes and strengthening that have completed a route by repeating the signal back to the route taken until it reaches the original source. This results in a route sending and receiving multiple signals signifying the fastest route. The Ant Colony Optimization also accounts for splitting routes as well as merging, as different routes may have similar paths towards the same destination. This also creates bias for the route to take as the receivers that accept the most signals given by different routes gain precedence.

### *C#*

According to Nakov and Kolev (2013), C# (pronounced as c-sharp) stands as a contemporary and versatile object-oriented, general-purpose programming language that Microsoft developed in conjunction with the.NET framework. It has evolved into a programming language of significant importance, finding widespread application across various domains. Within the expansive landscape of software development, C# has become instrumental in creating a diverse array of applications, spanning from office programs and desktop applications to web applications, websites, and mobile applications. Its versatility

lies in its ability to seamlessly integrate with the.NET platform, allowing developers to craft robust and scalable solutions for a multitude of scenarios. Comparatively, C# occupies a prominent position among high-level programming languages, showcasing similarities with well-established counterparts such as Java and C++. Additionally, it shares certain features with languages like Delphi, VB.NET, and C, contributing to its adaptability across different programming paradigms. C# has emerged as a linchpin in the programming landscape, evidenced by its status as one of the most widely used languages globally. Millions of developers leverage its capabilities, making it a cornerstone in software development practices worldwide.

In essence, C# embodies the essence of modern programming, offering developers a powerful and accessible language for creating a diverse range of applications. Its widespread adoption and continuous evolution underscore its significance in the ever-expanding realm of software development, making it a language of choice for developers seeking versatility, efficiency, and a robust framework for their projects.

### *Unity*

Unity, a game engine originating in 2005 and crafted by Unity Technologies, was a pioneering endeavor to democratize game development tools, a groundbreaking concept during its inception. Over its extensive lifespan, Unity has undergone substantial changes and expansions, adapting to contemporary practices and technologies within the game development industry. The engine's enduring focus is to equip the game development community with a robust set of tools while ensuring accessibility for developers of all skill

levels, including novices, aligning with its initial mission. (Zenva GameDev Academy, n.d.)

Beyond its significant impact in the gaming domain, Unity has made strides in real-time 3D development, underscoring its versatility as one of the most powerful engines available. The engine extends support for both 3D and 2D graphics, providing developers with the flexibility to choose their preferred art style. Unity's commitment to graphic diversity is reflected in its specialized tools, such as sprite sheet cutting for 2D graphics, and distinct script APIs catering to the physics options for each style. (Zenva GameDev Academy, n.d.)

Unity's architectural design also promotes transparency in composing game structures, where each level is segmented into scenes containing requisite game objects. The parent-child relationship feature facilitates the seamless addition of multiple objects, enhancing organization within the hierarchy. The Inspector tool provides quick access to object properties, offering developers the ability to make real-time adjustments without delving extensively into the code. (Zenva GameDev Academy, n.d.)

A standout feature of Unity is its scripting API, a powerful tool granting quick access to essential features and specific API calls that define the engine's functionalities. This API plays a pivotal role in defining Unity, underscoring its significance in shaping the engine's capabilities. Unity's cross-platform build support further distinguishes it, allowing developers to export games to an extensive array of platforms, from Android and iOS to Windows, MacOS, Linux, PS4, Xbox One, and more. The engine minimizes the need for extensive tweaks for different builds, streamlining the process of adapting games for diverse platforms. (Zenva GameDev Academy, n.d.)

Unity's vast Asset Store stands as a testament to its community-driven ethos, providing developers with a rich repository of graphical assets, game templates, audio, particle effects, and more. The store encompasses both paid and free assets, contributing to Unity's identity as a comprehensive platform. As Unity continues to evolve, its dedication to democratizing game development, fostering accessibility, and offering an extensive array of tools cements its status as a dynamic force in the gaming industry. (Zenva GameDev Academy, n.d.)

***Graphics Gale***

According to Sergeev (2017), GraphicsGale is a great tool for drawing and creating pixel art. Because of its many capabilities, the toolkit is a compelling choice for designers and artists who want to use creativity and efficiency in their pixel-based work. A wide range of file formats, including.ico,.gif,.avi,.cur, and.ani, are supported by GraphicsGale, guaranteeing compliance with various project specifications. The real-time animation preview function, which improves workflow consistency by enabling users to easily evaluate their animations while editing. Adding layers makes it easier to sketch and edit, and onion skinning gives you a useful visual reference by showing you the frames that came before and after.

A variety of processes are supported by GraphicsGale's adaptable export choices, which include sprite sheets and.gif files. The versatility of the toolkit is further enhanced with TWAIN imaging compatibility, which allows photos to be imported from a variety of devices, further enhancing creativity. In addition, elements unique to pixel art, such as color modification tools and palette management, facilitate the quick development of pixel art.

GraphicsGale is freely downloadable, highlighting its usability and encouraging artists to experiment with it and pursue their creative projects.

An article by ASOFTCLICK (2022), GraphicsGale is compared to a similar software Aseprite. Users love GraphicsGale for its easy-to-use animation capabilities, making it a popular graphics and animation program. The main benefit that sticks out is how simple it is to use, particularly for real-time animation preview. Users may easily refine their animations with the help of this function.

GraphicsGale has several options for creating and customizing art in addition to its animation features. These tools are useful and adaptable, allowing users to express their creativity. Having this variety of tools at one's disposal enhances the overall satisfaction of the user experience. The fact that GraphicsGale is free to use is a significant benefit. Because of its accessibility, it's a desirable choice for anyone looking for strong animation and graphics tools without having to pay extra money.

GraphicsGale is not without its shortcomings, though. One of its main drawbacks is that it can only be used with Windows, therefore Mac users cannot utilize it. Wine is a compatibility layer that lets Linux users run Windows applications on Linux PCs, therefore these users have discovered a solution. Although Wine works great with GraphicsGale, Linux users might find this dependence to be a drawback. With its user-friendly features, diverse creative tools, and free nature, GraphicsGale proves to be an impressive graphics and animation application. On the other hand, potential users should consider its platform limitation to Windows and the requirement for Wine for Linux users.

***Difficulty is Difficult: Designing for Hard Modes in Games***

The article points out that most, if not all video game difficulty modes are created right at the final development process where players and enemies alike are tuned for the sole purpose of balancing a game. And in order to create a difficult yet fair gameplay process for the player, the game must make the player feel liable for all mistakes present at the point of their defeat and never towards the mechanics present within the game. And one of the ways to do so is to give the player enough information whilst playing and after how and why they have lost. (Boutros, 2023)

The book acknowledges that the most straightforward way of increasing the difficulty within a game is to tweak damages received and invoked by the enemy. By reducing the damage an enemy takes while increasing their damage, the difficulty of a game will certainly be increased however without precise tuning of supplementary factors such as economy as well as enemy placement, the player will most-likely lose while making them feel helpless against the enemy. This sentiment of the player will also make them feel cheated out of an appropriately balanced difficult game.

Another known way to increase the difficulty of a game is to reduce the available safe and neutral zones for the player. This is achieved by replacing it with hazardous zones for the player or outright removing them by blocking them during the play through. This also gives an unintended pressure by pin-holing the player's attention on very few choices. A proper design of the map can leave the player pressured in a highly difficult situation.

Resource scarcity is also a popular way of intensifying the difficulty of a game. A popular example for a survival horror game is Resident Evil. Resident Evil gives the player

character a scarce number of bullets for its higher difficulty game mode. By doing so players are instead incentivized to use their knife to deal most damage as well as finishing the enemy. Games from other genres replicate this feeling by reducing the reward for doing certain tasks like defeating enemies. Making the economy of the game extremely poor at the start.

Increasing the Number of Enemies to fight against is also an obvious way to make games more difficult. This challenges players to observe, quickly think, and move under intense tension in order to quickly reduce the larger number of hordes of enemies attacking you. However, balancing achievability of such endeavor whilst also making it hard enough certainly requires a large amount of dedication from both the players, testers, and developers.

***Game Balance***

The book talks about the intricacies of how Developers Logically Balance a game through multiple layers each of which tips a metaphorical scale between Difficulty, Quantity, as well as Timing. Each of which changes the player's experience from the game but despite its importance especially for a Multiplayer game, a Game's Balance is context-sensitive and each player's performance and enjoyment are differently affected by a Game's Balance. (Schreiber & Romero, 2022)

The book also points out that a game's "Fun" is a completely different concept compared to a Game's Balance. A game might be completely balanced but not at all interesting and fun to play, but it is also possible that a game is completely unbalanced but most players will agree that the aforementioned game is very fun.

A game's balance also differs for the types of games a developer must adhere to. A common example is a game focused on a single player experience. It has completely different rules for balancing it compared to multiplayer games. Thus, developers must adhere to different kind of balance appropriate for their game. The book describes 7 different kinds of game balancing a developer uses.

The first kind of balance according to the book is about Mathematics-based Balance. This kind of balance uses Math to provide value to every element within a game's environment while using a spreadsheet to compare and contrast each element of the game. This kind of balance is usually used in cases where Precise numeral balance is required even as a player's playthrough progresses.

Another kind of balance is the Difficulty-based Balance also known as a Difficulty Curve. This notes the progression creates a proportional difficulty to keep the game challenging even as the game continues and the player gains more experience. This keeps the game challenging as the game progresses on and the game accumulates experience both literal as well as metaphysical with the growth of the player's avatar.

Progression-based balance, another kind of balance, refers to the rate of change a player goes through. From the Player avatar's Strengths, which increases the power of the player's avatar via the metaphysical experience, gaining abilities as they level up. The Difficulty Curve, giving the Player challenge by proportionally increasing the enemy strengths. The Gameplay Progression, which unlocks new modes, mechanics, item, maps, levels, to even ending the game. The moment a player says "This X has gotten long enough" shows a failure for the Gameplay progression. The last is the Narrative

progression, which shows the continuation of a game's story as the game progresses on or by reaching a certain threshold.

A kind of balance dedicated for a Multiplayer experience is the Balanced Initial Conditions. This refers to a state of the game, where during the start of a game, all players must have the exact same condition. Even if the team is the same or different, even if the team started with different characters. The starting conditions of all players must be the same. All advantages must be gained through the player's skill and knowledge with the game.

Another Multiplayer dedicated balance is Balance for Multiple Strategies. This allows for multiple builds, strategies, and selection, to prevent players from using the same builds, strategies, and selection over and over again. Certifying that all playstyles have valid strengths and counter play.

The Penultimate balance is the Balance between Game objects. This assures that each object available to the player has a reason for existing even if that situation is specific only to a certain event. This balance also assures that such objects are balanced compared to other objects. For example, if Object A gives +10 to damage for 500 gold and Object B gives only 5 Damage for 500 gold. Currently, Object B has no reason for existing for the game and must be balanced accordingly. If Object B instead has a cheaper price of 200 gold, then it will have a reason for existing. Another situation is if Both Object A and Object B gives +10 damage for the same value. This also creates a problem of Game Object Balance as again Object B has no reason for existing.

The Final Balance mentioned by the Book is Balance as Fairness. This balance exists due to a perceptual experience by the player. Assuming that the player's individual skill reaches a certain threshold, a balanced game should not give a player difficulty well outside of that range. If the game is a competition between players, then each player has a relatively equal skill between each other. If an in-game economy is balanced, then the costlier items must be more powerful compared to the cheaper ones.

According to the book, there are multiple modifiers to know if a game is balanced. First off is the game's difficulty. Difficulty encapsulates the entire experience of the game, asking only the question: is it too difficult, too easy, or exact? From every single interaction between players and enemy till the player has finished the game. The second factor is the game Quantity. This asks the question, is it too many, too little, or enough? The question encapsulates the resources the player gets, to the number of enemies a player encounter. Another factor is the game's Timing. This asks the question if any encounter within the game comes too early, too late, or within the right timing? Until all of these are tuned repeatedly, until a satisfactory answer for both players and developers arrives.

These modifiers however, are only the answers for each of the games core purposes. A game's purpose can determine if these modifiers are appropriate for both the players and developers. A common core purpose for a game is the Target Audience, if a game is created to target children as well as their parents, then giving them a game like Starcraft which is designed for serious players might be inappropriate. Another common core purpose is Designer intent, if a game is designed for playing to pass time while commuting, then it might be appropriate if the games are designed to be played in a casual sense, just a series of tasks to take people's mind off the commute. Another well-known common purpose for

a game is a game's goal, if a game is designed to be a resource management game, then it is necessary to give a limited supply of resources at least during the start of the game. And even if the game goes through the player must balance their economy so that one resource will be bloated while another is lacking.

A final aspect pertaining to game balance is progression. The players must feel that they are moving through their game specific goal even if only metaphorically. If within an RPG, they are fighting the exact same enemy with the exact same weapon with the exact same rewards over and over again after moving through multiple areas, then there will be a sense of monotony for the players playing the game and thus, players need to fight more challenging opponents with better weapons while receiving better rewards. In the end, a game's balance is only a metric on how both the players and developers feel towards it.

### *What is Game Balancing? An Examination of Concepts*

Balancing is widely seen as crucial to a game's success, but there is no agreed-upon definition of "game balancing." This study reviews fourteen publications by experienced game designers and other respected authors. A formal semantic analysis of these sources shows that no two authors share the same understanding of the term. The differing concepts are examined to identify key aspects, similarities, and differences. Interestingly, concepts like fairness, flow, and user satisfaction are important but not central to defining game balancing. (Becker & Görlich, 2020)

Effective game balancing is essential for creating successful games. However, there is a surprising lack of consensus on what "game balancing" means. In broad terms, it can be seen as tuning a game's rules, levels, difficulty, numbers, and algorithms to achieve

goals like keeping a game winnable, fair, challenging, and replayable. Much of the literature on game balancing comes from practitioners rather than scientists. Our review of fourteen publications by respected authors involved a formal concept analysis comparing their views on game balancing. The findings reveal diverse definitions, each focusing on different goals and aspects of game design. (Becker & Görlich, 2020)

Analysis shows that the fourteen authors have clearly different concepts of game balancing. They generally agree on some points, such as the importance of providing meaningful decision options to players. However, there is no single aspect or goal that all authors focus on. For example, Novak, one of the authors, views a balanced game as one that players perceive as consistent, fair, and fun. Schell, one of the authors, emphasizes keeping the player in a state of continuous flow, while Koster, one of the authors, focuses on providing a good level of challenge. These differences are partly due to some authors focusing on specific types of games, such as multiplayer games, while others address games in general. (Becker & Görlich, 2020)

The analysis also reveals that most authors focus on four key aspects of game balancing: characteristics of well-balanced games, difficulty, symmetry, and the balancing process itself. Well-balanced games should offer meaningful decisions, reflect player skill, and prevent dominant strategies. Most authors agree that finding the right level of difficulty is crucial and that it should increase with the player's skill. Fairness is inherent to symmetrical games but must also be ensured in asymmetrical games. However, fairness alone does not guarantee a well-balanced game. There are varying opinions on the balancing process and available methods, with some authors advocating for intentional imbalances or dynamic balancing.

In conclusion, the concept of game balancing remains diverse and lacks a universally accepted definition. While it is clear that balancing supports the design goal of fun, avoiding elements that hinder enjoyment, we still lack a positive, general definition of game balancing. Not all games are designed primarily for fun, and understanding what to avoid or how to tweak numbers is insufficient for a comprehensive game design plan. Further research is needed to develop a commonly agreeable definition and practical abstraction of the game balancing process. An empirical study on how good or bad game balance affects a game's success could highlight its importance and help identify proven properties, though this requires a deeper understanding than currently available.

***Game Balance: Learn to Find the Right Odds***

According to the article (Wirtz, 2023), developers make games balanced in order to create an exciting yet equal level playing field for the player. In games like Total War and Age of Empires, there are multiple different factions to choose from, each of them starts in a different location. All of them specialize in different units as well as giving some of their other units' minimal changes from each other. Some of them even use a resource unique only to them with a completely separate game mechanic, yet despite that, all of these promote the feeling of an equal level playing field.

The moment this delicate balance of different mechanics with different specializations tips over, is the start of a long and arduous problem faced by game developers. This process called balancing is the response of the developers to equalize the scales according to their vision. This either nerfs, or weakens certain skills, units, or

mechanics usually those overly used by their community, or buffing them, making them stronger. (Wirtz, 2023)

Despite the different mechanics and units faced by it, Games like Total War and Age of Empires still require some level of Symmetry especially for its starting conditions. While games focused on complete Asymmetry like Dead by Daylight do exist, where 1 Player competes against 4 Players with completely different game mechanics for both and are completely fine balance wise. Problem arises when Games that should be symmetrical but show apparent asymmetrical gameplay like Halo.

The Sniper rifle was infamous for pre-nerf in Halo 3. It was both overwhelmingly powerful compared to other weapons and it also fired quickly. This resulted in huge arguments amongst players and eventually Bungie, the game's developer, fixed this problem by significantly reducing the rate of fire of the gun. (Wirtz, 2023)

Total war games reduce this problem by introducing a certain level of Rock, Paper, Scissors amongst their gameplay. Cavalry is known for decimating against Archers, Archers easily defeats Infantry, and Infantry stops Cavalry dead in their tracks and wins against their Opponent's smaller unit size. Even for its Mechanics, some factions are great on one thing but usually lag behind on another. This Rock, Paper, and Scissors are even prevalent in Pokemon during the start, where the player battles against their rival. The Rival normally acquires the Pokemon that counters the player. And this forces the Player to Acquire the New Pokemons to defeat the Rival even against their starting Pokemon that counters your starting Pokemon.

***Level 16: Game Balance***

Game Balance alludes to keeping up harmony in a diversion to guarantee reasonable and agreeable gameplay. It's significant in both single-player and multiplayer games, including different viewpoints of reasonableness and challenge. In single-player games, balance involves fitting trouble to suit the audience's skill level. As players advance, the challenge regularly increments to coordinate their developing capability, known as pacing. Deciding the perfect challenge level includes broad playtesting, considering the differing abilities among players (Schreiber, 2009).

Multiplayer Games, particularly those with asymmetry, require balance to guarantee no starting condition offers an unjustifiable advantage. Indeed, symmetric recreations like Chess have subtleties, such as the first-move advantage, requiring cautious thought to preserve decency. Accomplishing adjustments in balance includes relating each player's assets to guarantee reasonable competition, regularly requiring arduous playtesting to fine-tune. (Schreiber, 2009).

Balancing Strategies for a game are significant when different ways to triumph exist. A prevailing procedure can dominate other alternatives, constraining gameplay differences. It's basic to guarantee that diverse techniques offer comparable chances of victory, cultivating strategic depth and player engagement. Playtesting plays a significant part in recognizing imbalances, highlighting which strategies are predominant or overwhelmed. (Schreiber, 2009).

In frameworks with different game objects like cards in trading-card games or weapons in role-playing games, balance involves guaranteeing each object's cost and

benefits adjust. This avoids any game object from being either very weak or very effective, advancing important player choices. Accomplishing balance in such frameworks regularly includes scientific modeling and arduous spreadsheet examination to evaluate and compare the viability of distinctive objects. (Schreiber, 2009).

Three essential strategies are utilized to balance game objects: Transitive Relationship, Intransitive Relationship, and Unique Distinctions. Transitive Relationships set up direct correlations between costs and benefits, rearranging balancing endeavors. In contrast, Intransitive Relationships make nuanced interactions where a few objects counter others, associated with rock-paper-scissors elements. Unique Distinctions make objects so different that coordinate comparisons are unreasonable, requiring broad playtesting to gauge effectiveness. (Schreiber, 2009).

Balancing Games regularly includes a mix of Mathematical Analysis, Creator Instinct, and Playtesting Input. Math makes a difference in setting up starting relationships between game elements, whereas creator instinct guides alterations to realize wanted gameplay encounters. Playtesting gives vital bits of knowledge into how players connected with the amusement, highlighting lopsided characteristics and zones for advancement. (Schreiber, 2009).

Balancing first-turn advantage in turn-based games is a common challenge, regularly tended to by pivoting beginning positions or giving compensatory assets. Furthermore, reporting lessons learned and rising methods guarantees nonstop improvement in balancing methodologies. Eventually, whereas balance is important, it's not the sole determinant of a game's success. A few recreations flourish in spite of awkward

nature, emphasizing the significance of imaginative opportunity and player delight over unbending adherence to adjust standards

### *Visual Studio Code*

According to Visual Studio Documentation (2021), Visual Studio Code (VS Code) is a user-friendly code editor that is well-known for its ease of use and efficiency in the programming process. Visual code facilitates development tasks such as debugging, task execution, and version management. Through extensions, Visual Studio Code supports a variety of programming languages, expanding its capability beyond its native JavaScript, TypeScript, and Node.js support. According to Visual Studio Frequently Asked Questions (2021), the emphasis is on providing the tools required for a quick code-build-debug cycle, leaving more complex workflows to comprehensive IDEs like Visual Studio. This editor, which is available for Windows, macOS, and Linux, supports JavaScript, TypeScript, and Node.js by default and has a wide variety of extensions for different languages and runtimes such as C++, C#, Java, Python, PHP, Go, and.NET.

### *Krita*

According to the Krita FAQ (n.d.), Krita is a free open-source application that is accessible on multiple platforms and offers a complete solution for creating digital art works. It's designed for frequent and concentrated use, with an emphasis on illustration, concept art, matte painting, textures, comics, and animations. Krita was created in collaboration with its users and caters to their individual needs and creative process. Furthermore, it is intended to conform to open standards and work in tandem with other software programs.

***Photoshop Reigns Supreme: How the Software Has Maintained Market Dominance***

Adobe Photoshop is a Photo Editing and Design Software created by Thomas and John Knoll in 1988. Adobe currently is the leading standard Photo Editing and Design Software for businesses across the world. Although it has a steep learning curve, Adobe Inc. provides many resources to help aspiring artists to create impressive digital art through different media such as books, website step-by-step guides, as well as internet videos and help improve their work better. Adobe Photoshop offers total creative control by giving its user the tool to customize photos, graphics, and more. It is also versatile as it can use photos, videos, and documents to fully utilize the user's skill. Finally, it has templates for its images, filter, background, and layers to jumpstart the editing process.

Adobe Photoshop's popularity amongst businesses is very warranted as businesses could utilize the photoshop to design detailed graphics and GIFs to publish in social media websites. Since Photoshop could also edit videos, it could also be used in Video Marketing as key material for self-promotion. It could also give seamless consistency for all products within a company by giving different projects the same font, palette, and art style to produce a professional effect for all your works (Peek, 2023).

***The Video Game Explosion: A History from Pong to Playstation and Beyond***

According to Wolf (2008), while the definition of Video Game has become loose over the years, it is normally defined as Virtual Games that gives control to the user via an input device such as Mouse, Keyboard, Controller, etc. and is typically expressed through an output visual device like a Monitor. However, although rare, there exists a few games that can facilitate blind, and other disabled individuals that would like to experience

playing video games. These are usually supplemented through both the game itself as well as using other devices specifically geared for their condition. Nonetheless, people enjoy the allure of games due to the entertainment they provide for the users especially amongst the younger generations of our time.

However, amongst the older generations, video games are seen as a curse as they see the bad influence of games on youth. One such popular account was about a game called Multi-User Dungeon (MUD), a text-based multiplayer online game released in 1978, the early days of the internet. This Game became so popular among youths especially the College undergraduates, that the United States' Military Satellite was having problems due to the massive number of users that was not built to handle. Eventually, the Military cut-off its services for MUD, only allowing service during the night. The Youths, not deterred from this situation, still played the game whenever the Game was available. This became so prevalent that the MUD started to have a nickname, Multi-Undergraduate Destroyer as it increased the number of absentees from the students who played it throughout the night.

Even throughout all of this, it should be said that video games are not without its benefits. Video Games are credited for reducing the stress of youths. Video Games are known for uniting various groups and ethnicities around the world. Some Games are focused on Tangential Learning, the ability of enjoyment through gaming while teaching concepts such as math and science. Games inspire the creativity of both youths and adults alike.

### *Rogue: Exploring the Dungeons of Doom*

Rogue, also recognized as "Rogue: Exploring the Dungeons of Doom," stands as a classic UNIX-Based video game innovatively crafted by Michael Toy and Glenn Wichman. Pioneering the integration of procedural generation, Rogue holds the distinction of being one of the earliest games to employ this technique, subsequently popularizing the terms "Roguelike" and "Rogue-lite" (Loguidice, 2009). The game leverages procedural generation to dynamically create map layouts, determine entry and exit points, place items, and position enemies. This groundbreaking approach imbues Rogue with remarkable replayability, injecting a sense of novelty into each playthrough by systematically randomizing its elements and providing players with fresh and unpredictable gaming experiences. Rogue remains a testament to the enduring impact of procedural generation in shaping the landscape of video game design.

### *Tower Defense Game based on 2D Grid Using Goal-Based Pathfinding Method*

Recent studies have ushered in a paradigm shift in the realm of tower defense games, a strategic genre where players strategically position units to impede enemy progress towards defined objectives. Traditionally, games in this genre lack autonomous path selection for in-game agents, featuring a multitude of enemies with varied behaviors converging on specific goals. These games traditionally rely on pathfinding methods to determine optimal routes between coordinates within an artificial intelligence movement system. (Sahuri, et al., 2023).

However, recent advancements have introduced a transformative objective-based pathfinding approach, revolutionizing gameplay by enabling player-initiated route

selection. This innovative technique facilitates dynamic in-game alterations as players modify destinations, compelling adversaries to dynamically adapt their course accordingly. Notably, this design also empowers players to strategically obstruct paths through unit placements, providing a means to manipulate available routes on the game map. These groundbreaking innovations introduce a more adaptive and strategic gameplay experience, reshaping the tower defense genre's dynamic landscape. In doing so, players are afforded greater control and a plethora of strategic possibilities, enhancing the overall gaming experience (Sahuri, et al., 2023).

***Adaptive Pathfinding and Utility AI Implementation for a Tower Defense Game***

This project focuses on creating a tower defense game with an advanced AI system to augment player challenge and enhance replay value. The core technique employed is the A* algorithm, serving as the foundation for an adaptive pathfinding system. This adaptive AI dynamically adjusts to the player's actions, demanding strategic thinking and action optimization during gameplay. In addition to pathfinding, an AI system for enemy spawning is developed, utilizing a Utility-based approach to introduce variability across replays and counter the player's actions. The comprehensive report encompasses game design, implementation of game mechanics, and the integration of the AI system using the Unity game engine. This initiative aims to create an engaging and dynamic gaming experience by leveraging sophisticated AI mechanisms within the tower defense genre. (Andre, 2022)

***Automatic Generation of Game Elements via Evolution***

The study by Ashlock (2010), "Automatic Generation of Game Elements via Evolution," explores the automatic production of puzzles using an evolutionary algorithm, aligning with the overarching theme of procedural generation in tower defense games. It introduces a system employing an evolutionary algorithm for puzzle generation, utilizing dynamic programming to optimize puzzles of varying complexities. The study delves into two puzzle types: chess mazes and chromatic puzzles, showcasing their potential for game content creation.

This study is relevant to the thesis, as both explore the usage of procedural generation. The Chess Maze Algorithm, a key component, similar to the thesis's focus on generating strategic paths in a tower defense game. Key concepts such as the Chess Maze Algorithm, dynamic programming, and implicit barriers are crucial for understanding how puzzles are optimized and generated within the evolutionary algorithm. The application of an evolutionary algorithm to generate puzzles, emphasizing the efficiency of the system in quickly producing diverse content. It recognizes the proof-of-concept nature and leaves details of puzzle integration to game designers, indicating a focus on adaptability. The study contributes to addressing the thesis's objectives by showcasing a systematic approach to puzzle generation. The Chess Maze Algorithm holds a significance for procedural path generation in tower defense games, ensuring solvability and strategic depth.

There are also potential challenges in puzzle simplicity and proposes multi-criterion optimization, distinguishing itself in the field. In conclusion, it establishes a foundation for procedural content generation, emphasizing adaptability and efficiency. The Chess Maze Algorithm and considerations for puzzle integration contribute valuable insights to the

broader theme of procedural generation. Future research directions include addressing puzzle simplicity concerns through multi-criterion optimization and exploring humaniform agents for evaluating puzzle difficulty. These suggestions offer avenues for enhancing the system's applicability in game design. The related study informs the thesis by providing a systematic approach to procedural puzzle generation. The Chess Maze Algorithm's relevance to tower defense path generation provides a potential solution with the thesis's goal of creating strategic and engaging content.

***Procedural Generation of Game-Worlds***

From the paper by McCooey (2021) titled "PROCEDURAL GENERATION OF GAME-WORLDS," delves into the realm of procedural generation techniques to create engaging environments for 2D platformer video games. In response to the growing demand for captivating video game experiences, algorithms investigated were specifically tailored for generating non-linear game-worlds. The research involves the implementation of a basic platformer game with fundamental goals and mechanics, such as player movement, jumping, and coin collection. Two distinct algorithms for generating game-worlds are explored, with the initial algorithm proving less satisfactory for the desired game outcomes. Subsequently, a maze-based algorithm is introduced, demonstrating improved customization for the creation of non-linear platformer game-worlds.

To assess the effectiveness of the procedural generation algorithm, this study conducted a user-based study where participants play the custom platformer game. The study evaluated two key criteria: the enjoyability of the game-world, determined through post-game survey responses, and the algorithm's efficacy in generating game-worlds at

intended difficulty levels. The game-worlds are generated with specific difficulty values, unknown to the players during gameplay. After completing each level, participants provided feedback on the perceived difficulty. Additionally, the study recorded various game statistics, including the time taken to complete the game, the player's success, and their proximity to failure.

The thesis contributed valuable insights into the practical application of procedural generation in the context of 2D platformer video games. Although the focus of the study is on a different genre (2D platformers), its relevance as a related study lies in the shared exploration of procedural generation techniques, emphasizing adaptability and customization. The study's methodologies, particularly the user-based evaluation, provided a framework that may inspire considerations for assessing the procedural content in the context of a 3D tower defense game. Overall, the work contributed to the broader understanding of procedural content generation and its impact on game experiences.

***Game AI Pro 360: Guide to Movement and Pathfinding***

Rabin (2020) talks about Pathfinding Architecture and how to optimize it as he infers that most developers should be informed about the topic as path requests are infamous for consuming a large percent of a computer’s resources especially in real-time strategy games and first-person shooters. The book states that A* (pronounced as A-star) is the favorite pathfinding search engine amongst developers, however, despite its popularity it has its flaws and details several techniques to enhance the speed and logic of the engine.

The book details different optimization techniques and one of which is to Build High-Quality Heuristics. It focuses on the typical programming trade-off of memory and speed and as such an example is precomputing every single path that the A.I. might take and store it for future use. This is stated to be the fastest path to generate at runtime however it uses a huge amount of memory. After the calculation, it suggests data compression to reduce the memory usage. One such way is manual compression which is known to work well, another way is by taking advantage of the environmental structure as usually in most maps, there are several optimal paths that are of considerable length. Most of these paths overlap at certain points and as such by placing transit nodes, redesigning the shortest path information would be easy, consuming less memory in the process

***Procedural Generation of Road Networks for Virtual Environments***

Martek (2012) explored the idea of using Procedural Generation on creating Road Networks for Virtual Environment to reduce both the intricacy and time loss on both world creation as well as the logistical challenges of logically connecting each part of the created virtual world with each other. The Author gives a promising solution for such problems, first, the planner designs the base tensor fields. Then, by applying the hyper streamlines of the tensor fields, the planner forms a street graph. And lastly, the formed graph is utilized to construct a three-dimensional model of the virtual world. This procedure forms both grid and radial patterns, as well as constructing roads that observes natural barriers such as trees and cliffs.

By utilizing a grammar-based, string rewriting system called L-System, Lindenmayer System or L-System is used to define a series of rules to alter a start string.

With this method, the L-System are utilized to form a general road template, which is then adjusted to adhere to the objective as well as logical city planning such as population density, and general overall patterns. A set of constraints is also applied to the template to make sure that no invalid roads are produced. This method can manage parameter modifications well, including changes to the overall pattern that directs the generation.

***Importance of User Interface and User Experience in Gaming.***

The Article teaches the significance of User Interface for the success of Video Games as players have multiple interactions towards the game's UI or User Interface especially since User Experience are only felt while User Interface are always viewed by the Player. A Great UI shows users what they are looking for while giving an appropriate visual experience thematic to the game they are playing. This enables the player to use the UI in its full capacity reducing frustration in interacting with the game. By reducing such frustrations, the game retains players who might have quit from a lackluster UI. A great UI also makes use of the available Hardware reliably by adjusting its size accordingly. (Ommzi Admin, 2018)

The Article has pointed out that many developers have avoided proper UI development due to Common Misconceptions about its development. The most common one is that UI development is a hindrance in Game Development. This is false since many interactions from the game uses UI for its interactions and as such deliberately slowing its development may cause more problems in the long run. Another Common Misconception is that proper UI development wastes precious developer time and money, one of the reasons given UI development is common sense, however this is false. Some UI are not

Intuitive and require delicate touch but most UI are easy and require no cost for its development. Not to mention that releasing a game without proper UI is the same as releasing it without play testing.

***A Novel Procedural Content Generation Algorithm for Tower Defense Games***

Based on the study of Dias et al., (2022), Tower defense games (TD) are a strategy gaming subgenre that focuses on defending territory by building defensive structures that block opponent progress. Map paths, opponent waves, defense towers, and multiple levels are all important features. The roguelike concept, which evolved from the 1980s game Rogue, focuses fundamental gameplay through randomly generated levels and items, with an emphasis on puzzle-solving, resource management, and replayability. This subgenre, inspired from roguelike games, incorporates tweaks to soften the punitive effect of permadeath, making it more acceptable to a larger audience while retaining characteristics of procedural generation and limited gameplay sessions.

The paper describes an algorithm that uses the Unity game engine to produce extensive levels for a roguelike tower defense game using minimalist voxel visual styles. The algorithm successfully produces maps, various opponent types, and coherent spawn sites that correspond to the expected gameplay experience. A playtesting session was conducted during the development process, which provided favorable feedback on procedural content generation and gameplay, allowing modifications to be made to improve gameplay, the overall game experience, and difficulty balancing.

***Procedural Content Generation for C++ Game Development***

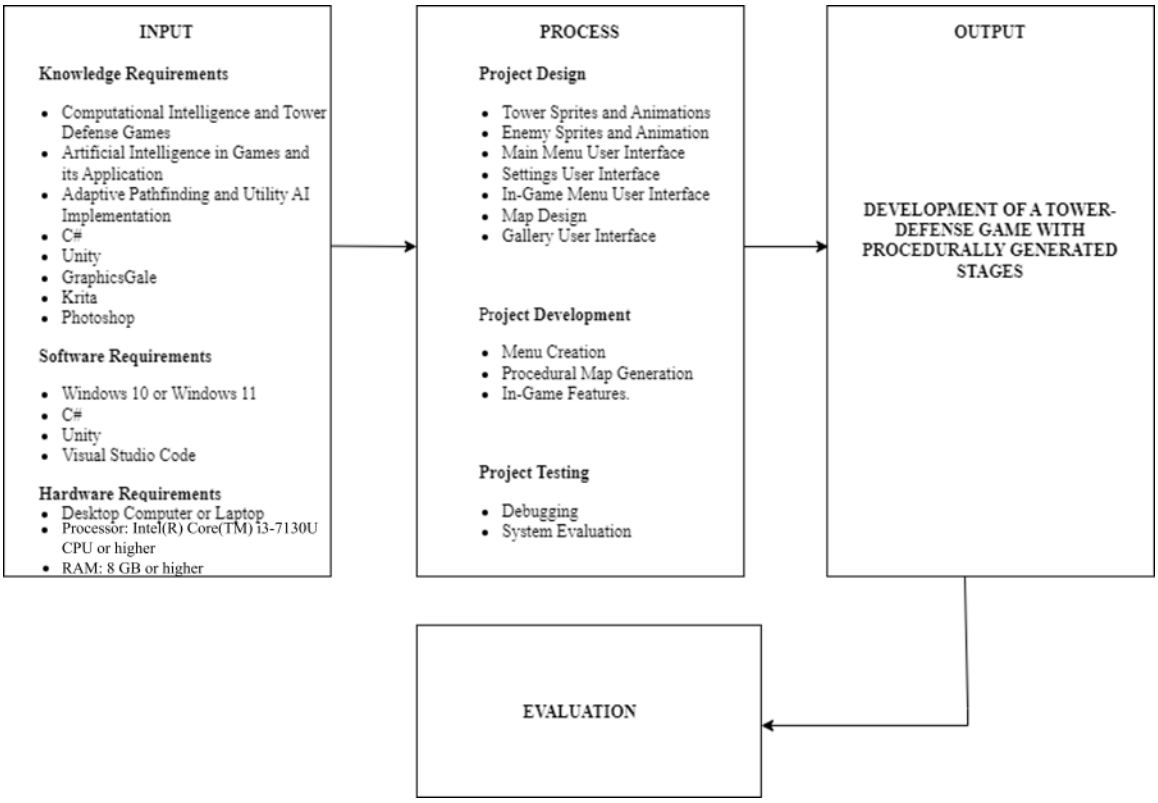
Procedural Generation and Random Generation are two distinct concepts. Procedural Generation is the process of creating concepts through an algorithm, while Random Generation is the ability to output pseudorandom results using an algorithm. The Book emphasizes that the result is only pseudorandom, not completely random as the computers are deterministic machines. Thus, by issuing a certain input should always output a certain result and in theory, the user could calculate the expected results whenever given a certain input.

The book also instructs the reader on how to generate random numbers in C++ as well as giving range to its randomness. After performing the instructions given by the book, it asks the reader to input 2 same numbers twice, it results in an equal result calling such value a Seed, proving a computer's deterministic trait. A Seed is the initial value an algorithm uses to produce procedural generated content. It explains that the intricacy of an algorithm is irrelevant. Utilizing the same value as well as the same operations, would create the same result. A Seed is very widely utilized in that games with procedural generation have the option of manually editing a seed before world generation. (Green, 2016)

## Conceptual Model of the Study

***Figure 1.***

Conceptual Model of the Study




The conceptual model shown in the figure above was constructed based on the findings, related literature, research presented, and insights. It illustrates and describes the essential elements needed for the system's development.

### Input

The input consists of the knowledge, software, and hardware requirements to perform the development of a tower defense game with a procedurally generated stage. The study requires knowledge about computational intelligence and tower defense games, artificial intelligence in games and its applications, adaptive pathfinding, and utility AI to develop the system. The system must also be developed using C# and Unity. A desktop

computer or laptop is required for the hardware requirements to completely develop the system. Researchers must also be familiar with and understand software testing and software quality evaluation to test the system.

**Process**

The process stage is composed of three components. It starts with the project design phase, where the necessary assets are created. Subsequently, project development is where the game itself is made; it involves creating the interface, stages, and in-game features. Finally, the phase of system testing addresses detected issues, offers solutions, and leverages insights from the project's development for future recommendations or iterations.

**Output**

The output of the study is the development of a Tower-Defense Game with a Procedurally Generated Stage using the Unity game engine and C# programming language. The output evaluation will involve gathering feedback from respondents using a purposive sampling technique. ISO 25010 will also be used to evaluate the output and define the quality of the software system. The evaluation aims to assess the game's functionality and maintainability based on the respondent's input.

**Operational Definition of Terms**

***Base*** refers to a central entity that functions as a defensive structure or point of defense. It frequently refers to a site or facility of strategic importance that offers protection or support for a range of operations.

***Boss*** refers to the enemy force that usually appears at the end of the level that has proportionally very high hit points compared to Mobs.

***Build Cost*** is the amount of Gold Required and Reduced to Build Towers

***Despawn*** is the ability to delete an entity from the play area to reduce Memory Usage.

***GameObject*** is an object in Unity that represents various game elements or components.

***Health Points*** is the amount of damage an entity can take before Despawning.

**Prefab** is a reusable and modifiable template for a GameObject in Unity.

***Procedural Generation*** is the Ability of an A.I. to continuously create digital objects.

***Procedurally Generated Conten*t** is an environment and/or entity that is created by Procedural Generation.

***Props*** refers to objects or items in the game environment that are not part of the core gameplay mechanics but more so for visual aesthetics.

***Roguelike*** is a Video Game Genre that randomizes a lot of its mechanics by using Procedural Generation in order to create different experiences for the player.

***Rogue-lite*** is a Subgenre of Roguelike games that enables progression after a player character death or game over. The Progression is usually caused by giving a player some currency after game-over that can be spent to upgrade or unlock new features.

***Scenes*** an environment within a Unity project where various components are presented to the user.

***Spawn*** is the ability to generate a Predetermined Character within the play area.

***Spawn Location*** is the location where Mobs Spawn.

***Tower*** is an entity whose job is to attack enemies.

***Tower Defense*** is a type of Video Game that focuses on Defending a Base from Enemies by building Towers.

***Gold*** is the amount of Game Currency the Player has accumulated. These Points can be spent to build and upgrade Towers.

***Upgrade*** is the ability to strengthen Player's Structures.

***Upgrade Cost*** is the amount of Tower Points required to Upgrade Towers.

# Chapter 3

# METHODOLOGY

This chapter contains the project design, development procedures, testing processes, and evaluation procedure of the game system.

## Project Design

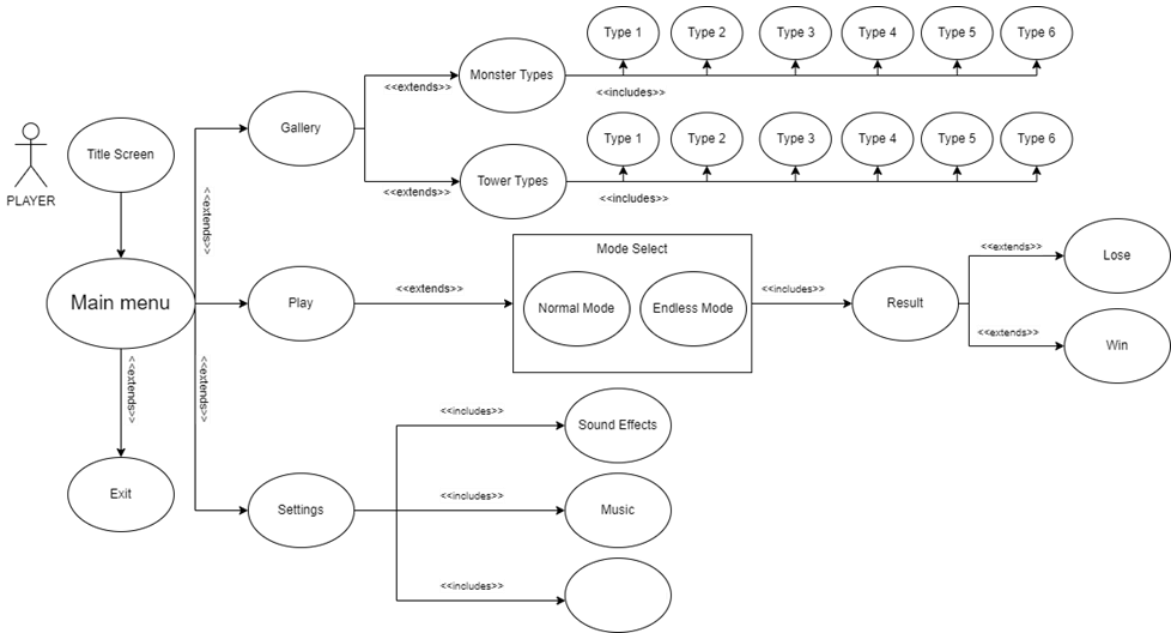
### *Use Case Diagrams*

The core tower defense game mechanics follow the conventional elements of players defending against waves of enemies by strategically placing defensive towers and upgrading them. Procedural generation allows for dynamically generated layouts and varied structure of game levels, ensuring a unique gameplay session on each playthrough. The pathfinding algorithm being randomized without any consideration of the shortest path will take away from the traditional linear enemy pathfinding, allowing for more unpredictable and encouraging players to adapt.

The pathfinding system and procedural level generating modules will be part of the technical architecture. Using procedural generation methods to dynamically construct terrain and paths is the implementation strategy. The game's replay value will be increased by the development of a pathfinding algorithm that will provide a variety of paths with varying degrees of complexity and difficulty.

***Figure 2.***

Index Use Case Diagram




The Index Use Case shown in Figure 2 is the overview of the processes within the tower defense game. The game starts at the Title Screen which prompts the player to click the screen. Then the player will be presented with the Game Menu which includes Play button, Gallery button, Settings button, Exit button.

The Play button proceeds to Mode Select which prompts the player to choose between Normal or Endless Mode. In this phase, the game will generate the stages utilizing procedural content generation. Then the process diverges into either of the two possible outcomes: if the player runs out of health, the player will be sent back to the game menu to start again or exit the game. The Gallery button allows the player to view the Enemy Type and Tower Types which includes their Lore, Health Points (HP), and Damage. The settings button allows the player to adjust the Music Volume and Sound Effects Volume with the use of a volume slide. It also includes the Resolution button that allows the player to choose

the screen resolution for the game. Finally, the Exit button which allows the player to exit or quit the game.

**Table 1.**

*Use Case Titled and Description of the Index Use Case*

| Use Case No. | Use Case Title | Use Case Description |
|---|---|---|
| UC01 | Title Screen | The case begins when the application is started. This scenario displays the general synopsis of the game and the prompt to click to go to the main menu. |
| UC02 | Main Menu | This case gives the user access to all the game's other cases and commands. The main menu has the Mode Select, Settings, Gallery, and Exit buttons. |

| | | |
|---|---|---|
| UC03 | Mode Select | This case is where the player will choose the Game mode they want to play. |
| UC04 | Normal Mode | This case is where the player needs to clear a set of waves to clear the game. |
| UC05 | Endless Mode | This case is where the game will only end when the player dies. |
| UC06 | Settings | This case is where the player can adjust the Music and Sound Effects Volumes with a slider. |
| UC07 | Gallery | This case is including the information about the Tower and Enemy types are |
| UC08 | Exit | This case terminates the application. |

***Block Diagram***

***Figure 3.***

Block Diagram

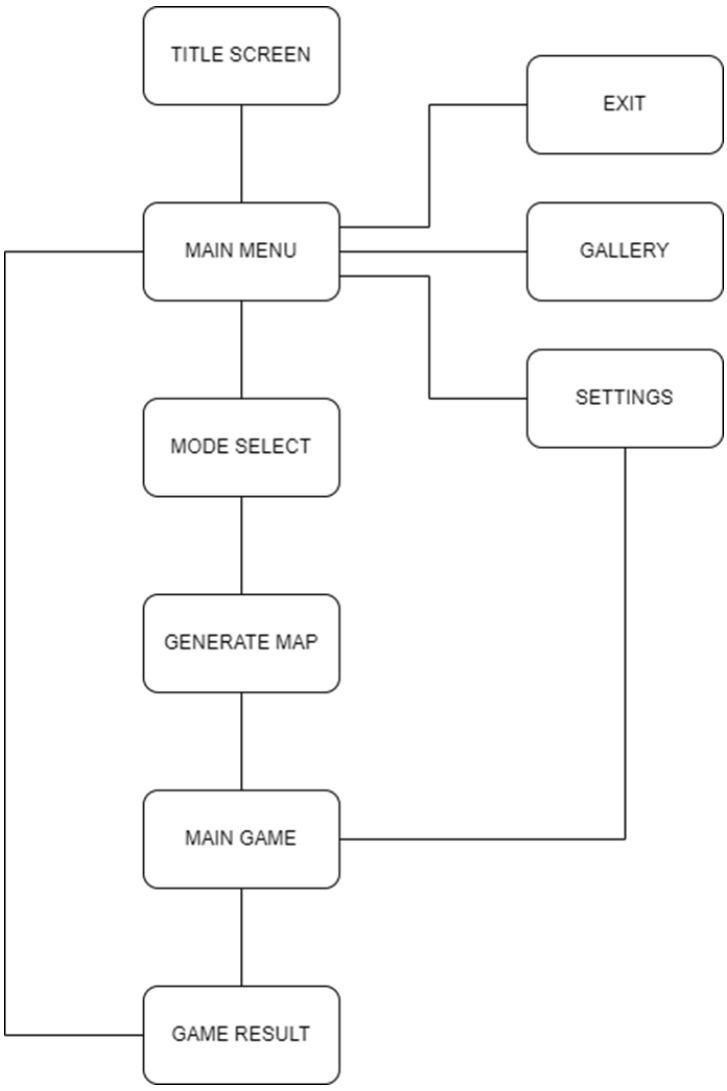



Figure 3 shows a Block Diagram of the game's general system design. It provided an overview of the game's flow and architecture.

***Low Level Design***

***Figure 4.***

Low-Level Design

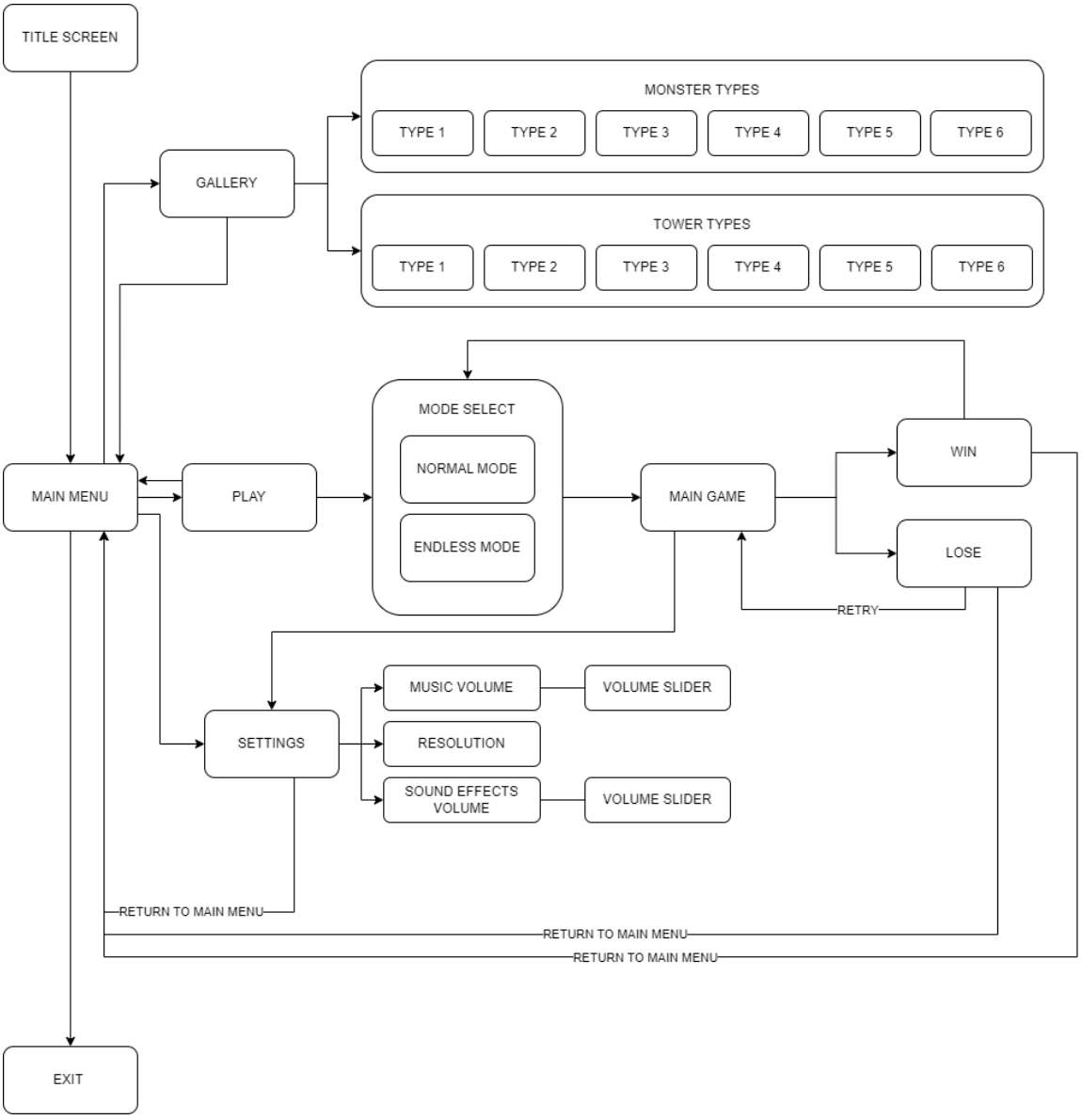
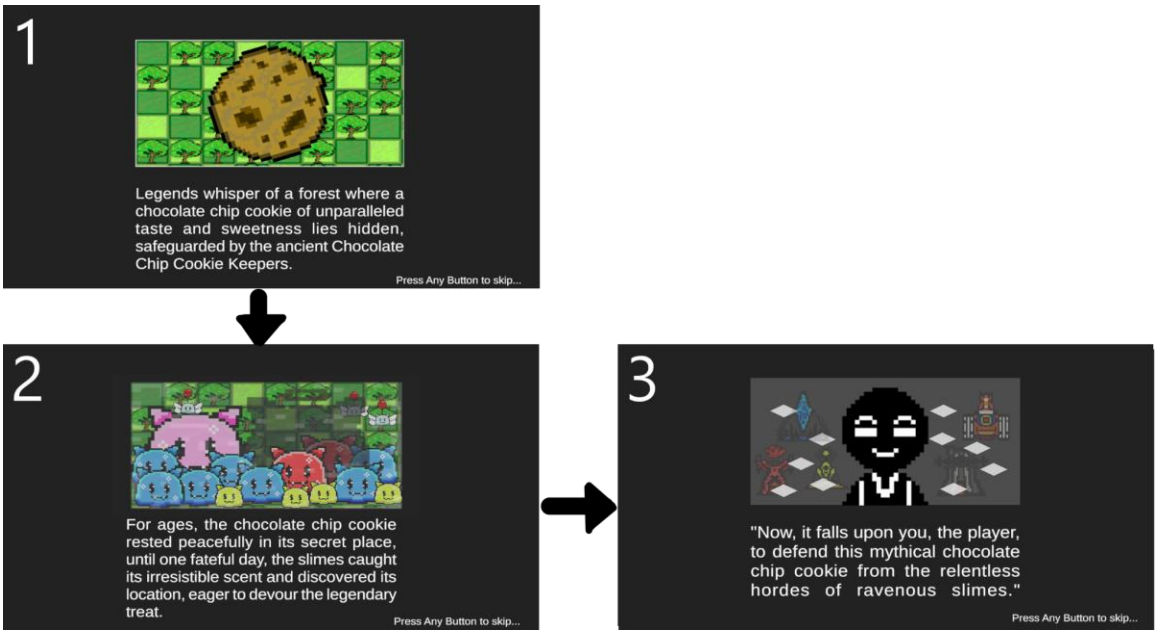



Figure 4 shows the Low-Level Design (LLD) for the implementation of the High-Level Design (HLD) numerous components and phases on a specific level.

**Storyboard**

***Figure 5.***

Storyboard




The game's premise is shown in Figure 5. There is a legend of a cookie with unparalleled taste and sweetness hidden in a forest safeguarded by ancient chocolate chip cookie keepers. For ages, the cookie rested peacefully in its secret place, until one fateful day, the slimes caught its irresistible scent and discovered its location, eager to devour the legendary treat. It's up to the player to defend the chocolate chip cookie from the hordes of slimes.

**Wireframe**

***Figure 6.***

Wireframes

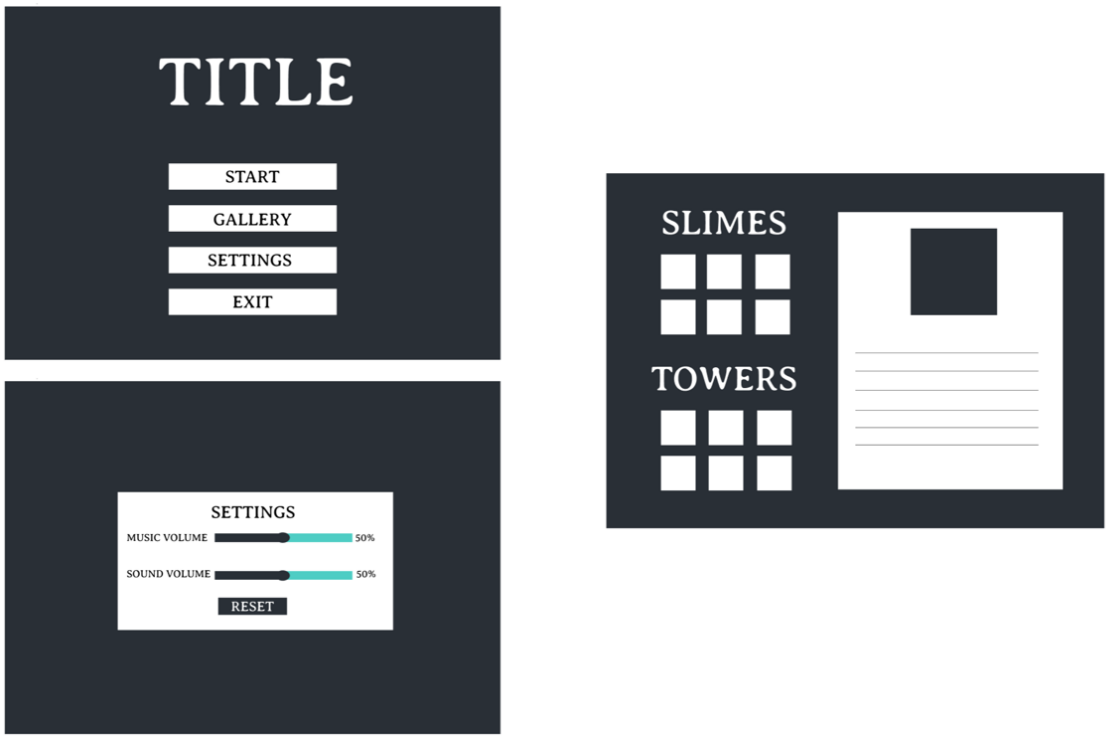


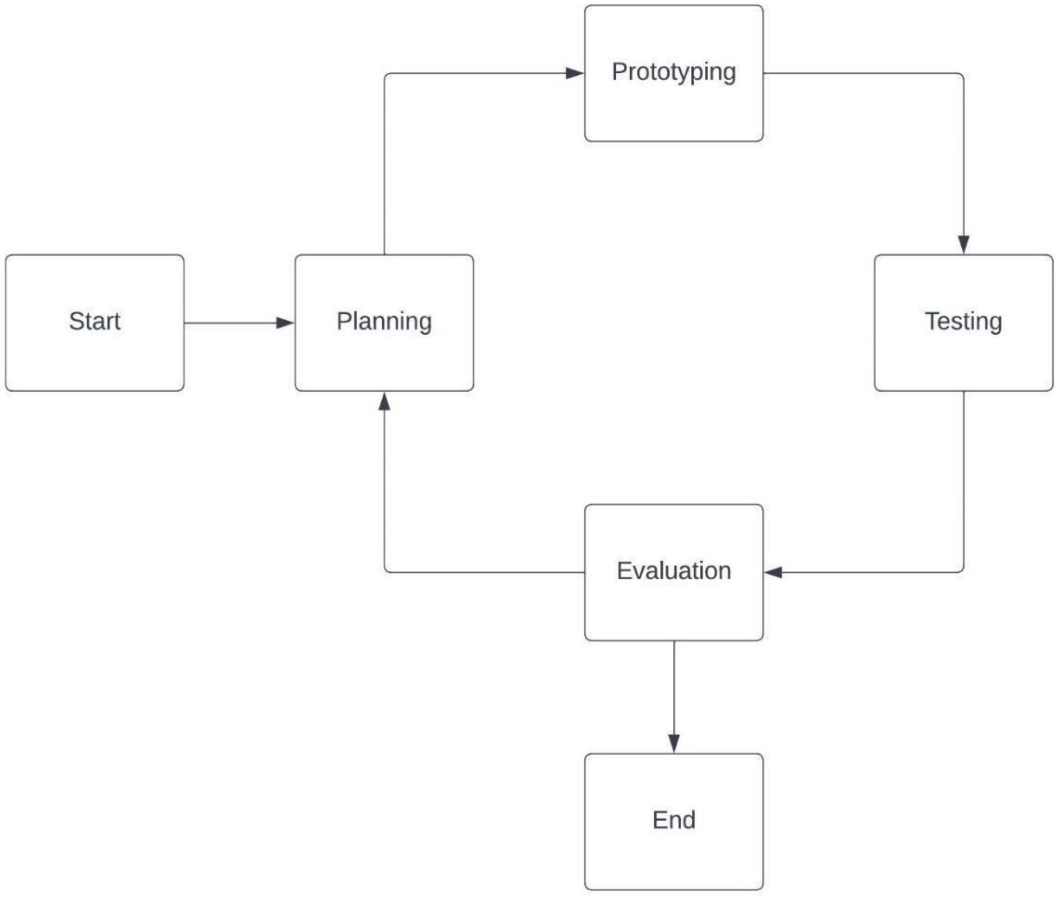

Figure 6 shows the wireframes for the desired User Interface of the Procedural Tower Defense Game. These are the Main Menu, Gallery, the Settings, and information about Slimes and Tower.

**Project Development**

To produce an engaging and enjoyable game, the Tower Defense game will be built using the Unity Game Engine. The Rapid Application Development methodology will be utilized to shorten the development process and create a dynamic and responsive system. This process enables adaptation to changing needs, feature iteration, and the delivery of the finished Tower Defense Game.

***Figure 7.***

Rapid Application Development Model




The figure above shows the process of the Rapid Application Development (RAD) Model from the start to the end of the development cycle.

**Planning**

The Planning phase begins by Identifying the Objectives and Defining Scopes and Delimitations. Later, as the Evaluation phase comes back to the Planning phase, it adjusts the Project based on the assessment generated from the Evaluation phase.

**Prototyping**

The Prototyping phase creates the device based on the Suggested Ideas on the Planning Phase. After a cycle, the Prototyping phase normally adjusts the device created during the previous cycle but it may start from the beginning if the Planning phase deems it accordingly.

**Testing Phase**

The Testing phase conducts systematic analysis on the created device and writes the results from each cycle. It also notes the Identified Problems observed during the analysis as well as clearing out any Problems resolved during each cycle.

**Evaluation Phase**

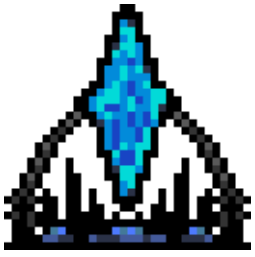
The evaluation phase shows alignment of the application with the project requirements following the conclusion of the testing phase. In case the playtesting phase reveals inadequacies or suggests additional enhancements, the researchers will devise additional potential solutions for the identified issues.

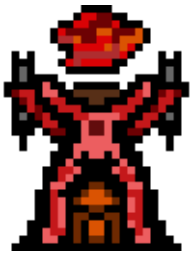
***Game Elements***

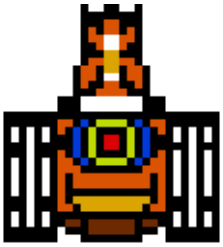
There are several tower types which can be built by the player to protect the base by attacking enemies within its attack range. Each tower having its own unique characteristic

**Table 2.**

*Tower Types and Characteristics*

| | Tower Type | Description | Characteristics | Stats Values |
|---|---|---|---|---|
|  | Basic Tower | Basic single target tower. | Basic single target tower with moderate damage, attack speed, and range. | Attack: 10<br>Cooldown: 2.5 secs<br>Upgrade Cost:<br>Range: 2.5 units<br>Build Cost: 10G |
|  | Ice Tower | Slowing Tower | Slowing Tower. Attacks inflict a slow effect which reduces movement speed for enemies that it | Attack: 3<br>Cooldown: 2<br>Range: 2 units<br>Build Cost: 15G |

| | | | | |
|---|---|---|---|---|
| | | | hits. Attacks deal low damage and have a relatively short range. | |
|  | Fire Tower | Splash Damage Tower | Attacks inflict damage on its target and enemies near it.<br>Slow Attack Speed<br>Has a long attack range | Attack: 15<br>Cooldown: 5sec<br>Range: 3 units<br>Build Cost: 30G |
|  | Shock Tower | Fast attacking Single Target Tower. | High Damage Per Second but Low Base Damage<br>Fast Attack Speed | Attack: 5<br>Cooldown: 0.5secs<br>Range: 1 unit |

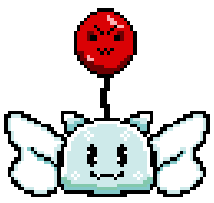
| | | | | |
|---|---|---|---|---|
| | | | Short Attack Range | Build Cost: 25G |
|  | Mega Tower | High Damage Long Range Single Target Tower | A powerful tower with high damage and long range but takes longer to reload attacks. | Attack: 50 Cooldown Upgrade Cost: Range: Build Cost: |

Several enemy types are created to actively target the player's bases. Each monster type has its own characteristics to differentiate from one another.

**Table 3.**

*Monster Types and Characteristics*

| | Type | Description | Characteristics |
|---|---|---|---|
|  | Blue Slime | A simple-minded gelatinous slime monster, it is generally harmless on its own. | • 0.75 Move Speed<br>• 30 Health |

| | | | |
|---|---|---|---|
|  | Yellow Slime | A gelatinous slime monster, its body seems to be quite slippery thus allowing it to move fast. | • 2 Move Speed<br>• 20 Health<br>• Fast. |
|  | Pink Slime | A gelatinous slime monster, it is able to absorb more damage due to its elastic and bouncy body. | • 0.5 Move Speed<br>• 150 Health. |
|  | Red Slime | A gelatinous slime monster that is highly active and might explode to multiple tiny versions of itself. | • 0.5 Move Speed<br>• 75 Health<br>• Splits into 3 Lesser Red Slimes |
|  | Flying Slime | A gelatinous slime monster that has wings. It actually cannot fly on its own | • 0.75 Move Speed.<br>• 20 Health. |

| | | | |
|---|---|---|---|
| | | and relies on the balloon attached to it. | • Flying. |
|  | Tako Slime | A gelatinous slime monster that vaguely resembles an octopus with very short tentacles. Its body is quite resilient. These slimes are particularly fond of cookies, word puns and dad jokes. | • 0.25 Move Speed.<br>• 420 Health. |

**Pixel Art Sprites Creation**

***Figure 8.***

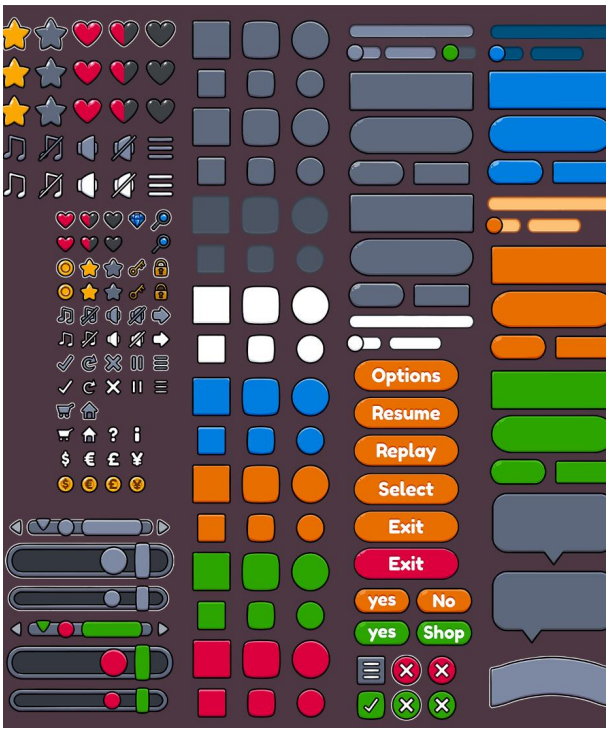
Towers, Projectiles, Slimes, Tiles, and Props Sprite Sheet

The figure shown above is the various art components created using GraphicsGale for the towers, projectiles, slime monsters, props (trees, rocks, cookie, spawn point), and tiles used for the terrain.

**Game User Interface**

***Figure 9***.

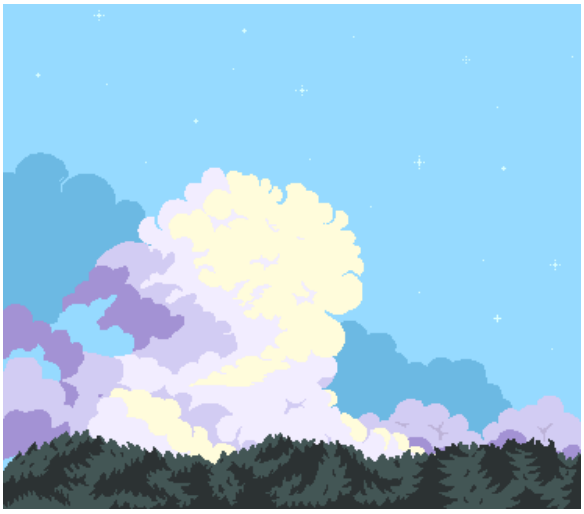
Penzilla's Basic GUI Bundle




The figure shown below presents the user interface assets utilized in the making of Slimes Raid Cookie TD. The game utilized a simple and minimalistic theme for the overall graphical user interface created by Penzilla.

***Figure 10***

samvieten's pixel art painting of clouds.



The figure above shows the background image used in the main menu of the game. It is a pixel art painting of clouds made by samvieten uploaded on their page on itch.io.

**Game Music and Sound Effects**

The audio sound effects used in the development of Slimes Raid Cookie TD were free to use sound effects uploaded by YouTube channels SoundX, Sandra Michelle, Youtube Sound Effects, 2MirrorsDialogue, ZapSplat Sound Effects, SoundMaster, Alexis KB Sounds. Respectively titled "Laser Beam Cannon - Sound Effect | Free Sound Effect", "Electricity Sound Effect xd (Thx for 1M Views!)", "splat sound effects (no copyright)", "Stone Hit Sound Effect", "Fireball Explosion Sound Effect", "Ice Sound Effects", and "Ice Crack Freeze Sound Effect". The three pieces of music used in this project is obtained from the DOVA-SYNDROME Official Youtube Channel titled "クレイジークッキング" by T.Waraya and "I hope it will be fine tomorrow" by shimtone, and then "Free 8-Bit Mystery Theme Music For Video Game Development" uploaded by Keith6535 on their

YouTube Channel. The soundtracks are publicly available and free of use for commercial and non-commercial purposes.

**Game Development**

The proponents used Unity Game Engine and Visual Studio Code as their main tools for programming and executing the features of the tower defense game. Here are the key scripts employed in Slimes Raid Cookie TD:

- **AgentBehavior Script:** This script was employed to dictate the behavior of the enemy slime monsters, encompassing pathfinding, slowing effects, and changes in frozen debuffs.
- **SlimeSpawner Script:** It determined the spawning mechanics of enemy slime monsters, specifying spawn intervals and the quantity per wave.
- **Stats Script:** This script stored essential information such as health, earned gold upon destruction, and multipliers that augment health and gold earnings with wave progression.
- **GridGeneration Script:** Responsible for instantiating grid tiles and relevant node information.
- **PropsPlacement Script:** Utilized Perlin Noise algorithm to generate obstacles like trees and rocks on the grid.
- **TowerAI Scripts:** Governed the targeting logic of towers, focusing on the nearest enemy within their attack range, and managed relevant tower stats for projectile interactions.

- **ProjectileAI Scripts:** Controlled the movement and behavior of projectiles launched by towers, including collision logic with enemy targets.
- **MouseClickSystem Script:** Handled mouse click interactions during gameplay, such as tower placement, selection, upgrading/selling, and prop removal.
- **Node Script:** Stored pertinent information within grid nodes, such as coordinates, walkability, buildability, and pheromone levels.

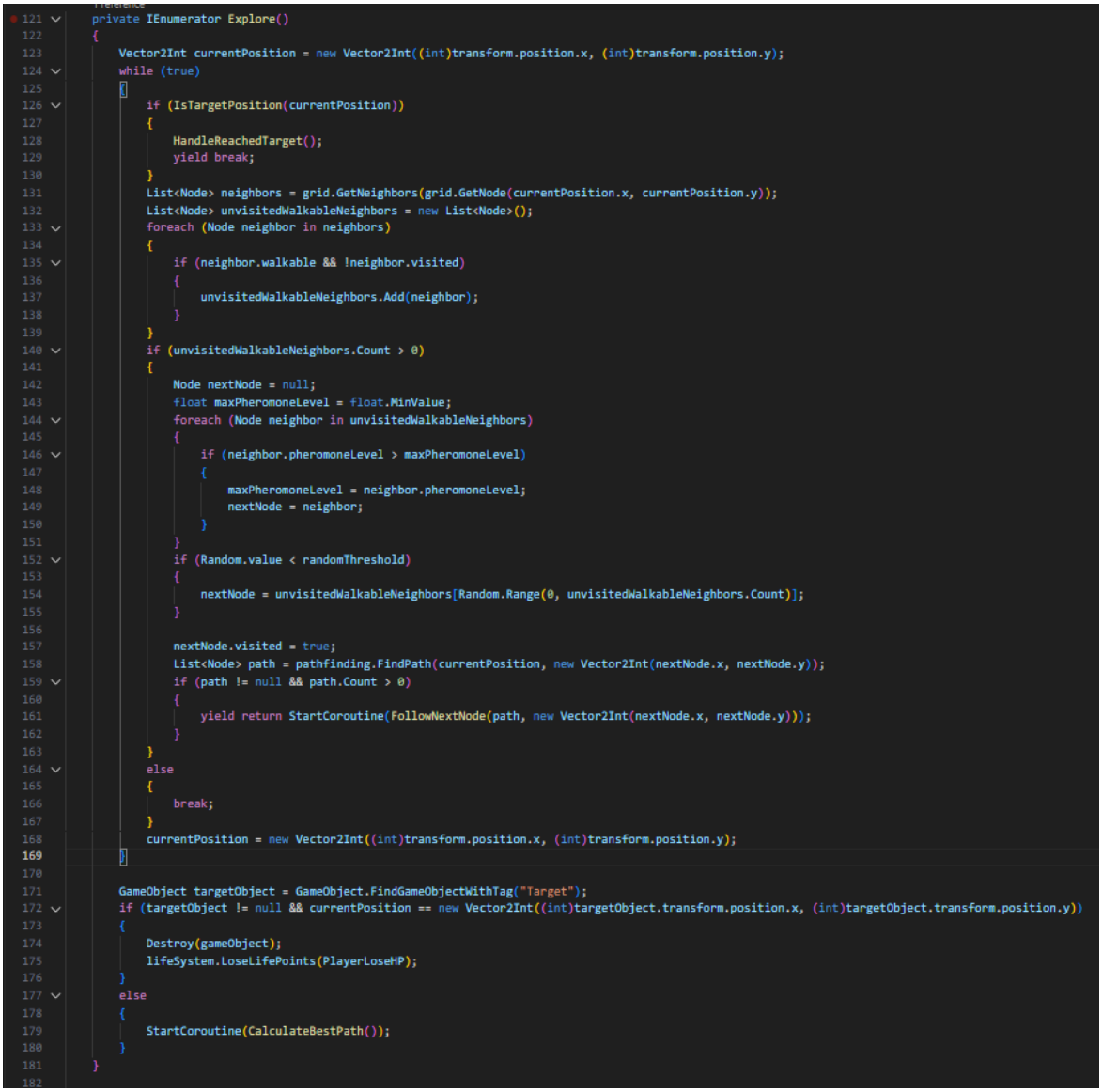
**Implementation of the Ant Colony Optimization for the Enemy AI**

The Explore() method in the Agent Behavior Script is a coroutine that simulates the behavior of an agent exploring its environment using principles inspired by Ant Colony Optimization (ACO). The agent navigates a grid-based environment, seeking to reach a target position while considering the pheromone levels and walkability of the nodes it visits. The method iteratively updates the agent's position, making decisions based on the pheromone levels and random factors to simulate natural exploration behavior.

It is responsible for controlling the agent's exploration behavior. The agent starts at its current position and repeatedly selects the next node to move based on the pheromone levels of its neighboring nodes. The agent aims to find and reach a target position within the grid. If the agent encounters the target or exhausts all possible moves, it executes the corresponding actions.

***Figure 11.***

Explore Method that Simulate ACO System



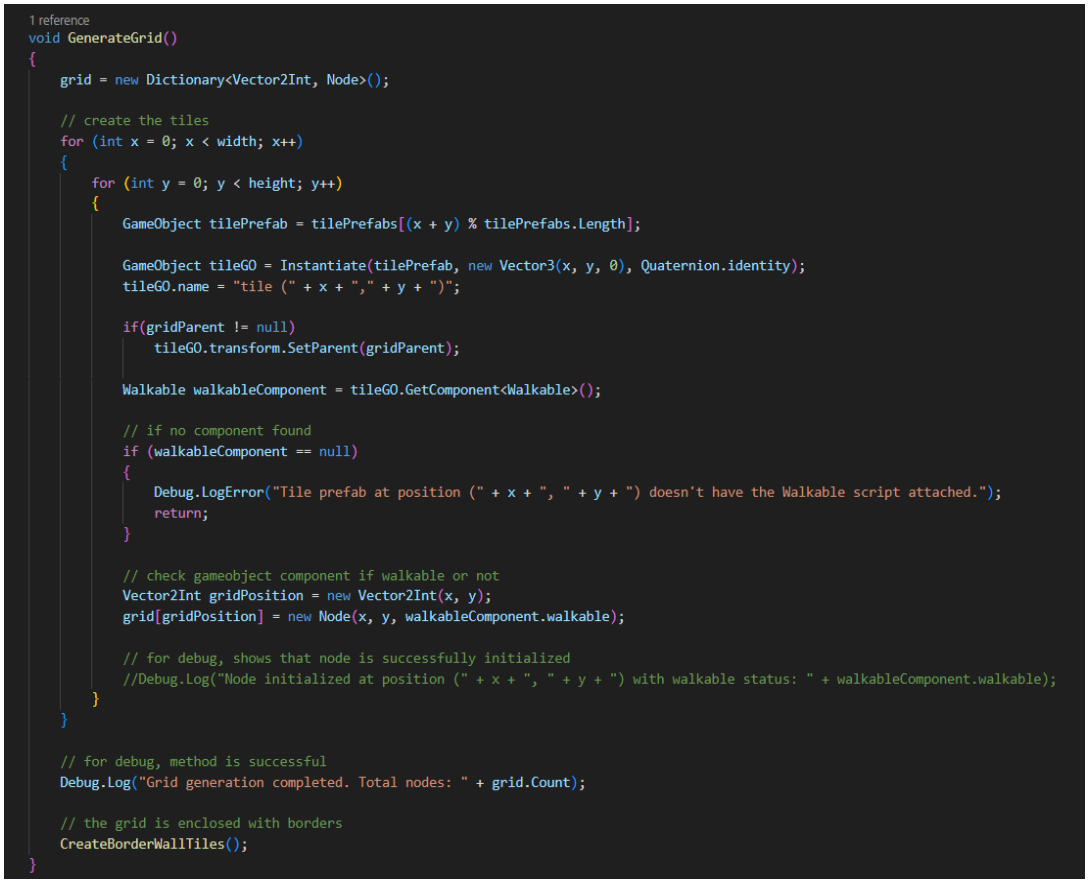
```
private IEnumerator Explore()
{
    Vector2Int currentPosition = new Vector2Int((int)transform.position.x, (int)transform.position.y);
    while (true)
    {
        if (IsTargetPosition(currentPosition))
        {
            HandleReachedTarget();
            yield break;
        }
        List<Node> neighbors = grid.GetNeighbors(grid.GetNode(currentPosition.x, currentPosition.y));
        List<Node> unvisitedWalkableNeighbors = new List<Node>();
        foreach (Node neighbor in neighbors)
        {
            if (neighbor.walkable && !neighbor.visited)
            {
                unvisitedWalkableNeighbors.Add(neighbor);
            }
        }
        if (unvisitedWalkableNeighbors.Count > 0)
        {
            Node nextNode = null;
            float maxPheromoneLevel = float.MinValue;
            foreach (Node neighbor in unvisitedWalkableNeighbors)
            {
                if (neighbor.pheromoneLevel > maxPheromoneLevel)
                {
                    maxPheromoneLevel = neighbor.pheromoneLevel;
                    nextNode = neighbor;
                }
            }
            if (Random.value < randomThreshold)
            {
                nextNode = unvisitedWalkableNeighbors[Random.Range(0, unvisitedWalkableNeighbors.Count)];
            }

            nextNode.visited = true;
            List<Node> path = pathfinding.FindPath(currentPosition, new Vector2Int(nextNode.x, nextNode.y));
            if (path != null && path.Count > 0)
            {
                yield return StartCoroutine(FollowNextNode(path, new Vector2Int(nextNode.x, nextNode.y)));
            }
        }
        else
        {
            break;
        }
        currentPosition = new Vector2Int((int)transform.position.x, (int)transform.position.y);
    }

    GameObject targetObject = GameObject.FindGameObjectWithTag("Target");
    if (targetObject != null && currentPosition == new Vector2Int((int)targetObject.transform.position.x, (int)targetObject.transform.position.y))
    {
        Destroy(gameObject);
        lifeSystem.LoseLifePoints(PlayerLoseHP);
    }
    else
    {
        StartCoroutine(CalculateBestPath());
    }
}
```

**Implementation on the Procedural Generation**

In the following code snippets from the Game Grid, Perlin Noise, and Spawn Point scripts, the procedural generation in the project's tower defense game is shown. From the grid generation that initialize the nodes and instantiates the floor tiles, moving the core object (the cookie), instantiating a 3x3 structure in the center of the grid, instantiating props (trees and rocks) randomly over the grid, instantiating the spawn points, and laying down the path tiles connecting the spawn points and the center of the playable grid area.

***Figure 12.***

Generate Grid Method



```
1 reference
void GenerateGrid()
{
    grid = new Dictionary<Vector2Int, Node>();

    // create the tiles
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            GameObject tilePrefab = tilePrefabs[(x + y) % tilePrefabs.Length];

            GameObject tileGO = Instantiate(tilePrefab, new Vector3(x, y, 0), Quaternion.identity);
            tileGO.name = "tile (" + x + "," + y + ")";

            if(gridParent != null)
                tileGO.transform.SetParent(gridParent);

            Walkable walkableComponent = tileGO.GetComponent<Walkable>();

            // if no component found
            if (walkableComponent == null)
            {
                Debug.LogError("Tile prefab at position (" + x + ", " + y + ") doesn't have the Walkable script attached.");
                return;
            }

            // check gameobject component if walkable or not
            Vector2Int gridPosition = new Vector2Int(x, y);
            grid[gridPosition] = new Node(x, y, walkableComponent.walkable);

            // for debug, shows that node is successfully initialized
            //Debug.Log("Node initialized at position (" + x + ", " + y + ") with walkable status: " + walkableComponent.walkable);
        }
    }

    // for debug, method is successful
    Debug.Log("Grid generation completed. Total nodes: " + grid.Count);

    // the grid is enclosed with borders
    CreateBorderWallTiles();
}
```

In the Generate Grid Method in the Game Grid script, the script initializes the grid and the nodes to be used for various game features. The visualization of the nodes are shown as tiles placed into a checkered chessboard pattern and then calls the method that creates the walls of the playable area.

***Figure 13.***

Create Border Wall Tiles method

```
1 reference
void CreateBorderWallTiles()
{
    for (int x = -borderWidth; x < width + borderWidth; x++)
    {
        for (int y = -borderWidth; y < height + borderWidth; y++)
        {
            // prevents the walls being instantiated inside the playable grid
            if (x < 0 || x >= width || y < 0 || y >= height)
            {
                Vector3 position = new Vector3(x, y, 0);
                GameObject wallTileGO = Instantiate(wallTilePrefab, position, Quaternion.identity);
                wallTileGO.name = "wall (" + x + "," + y + ")";

                if(gridParent != null)
                    wallTileGO.transform.SetParent(gridParent);
            }
        }
    }
}
```

In the Create Border Wall Tiles method in the Game Grid script, it generates the walls outside the specified height and width of the game's grid by visualizing them as wall tiles.

***Figure 14.***

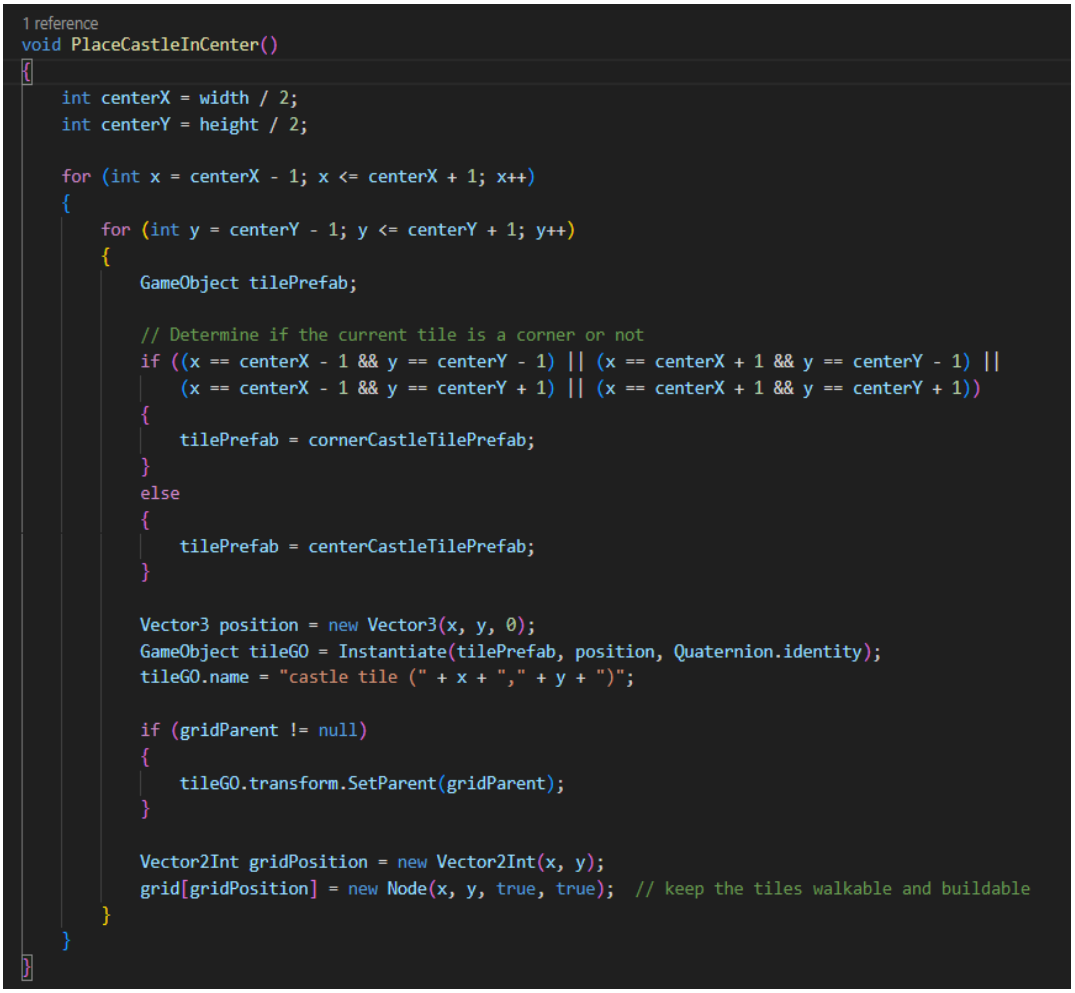
Move Core To Center method

```
//simply moves the "core" game object to the center of the grid
1 reference
void MoveCoreToCenter(int width, int height)
{
    int centerX = width / 2;
    int centerY = height / 2;
    Vector3 centerPosition = new Vector3(centerX, centerY, 0);

    if (Core != null)
    {
        Core.transform.position = centerPosition;
        UpdateNodeWalkability();
        UpdateNodeBuildability();
    }
    else
    {
        Debug.LogError("Core object is not assigned.");
    }
}
```

In the Move Core to Center method in the Game Grid script, the core game object is moved from a different location into the center of the grid. The core object in the context of the game is the Cookie that the player needs to defend which is also where other features are centralized on.

***Figure 15.***

Place Castle In Center method

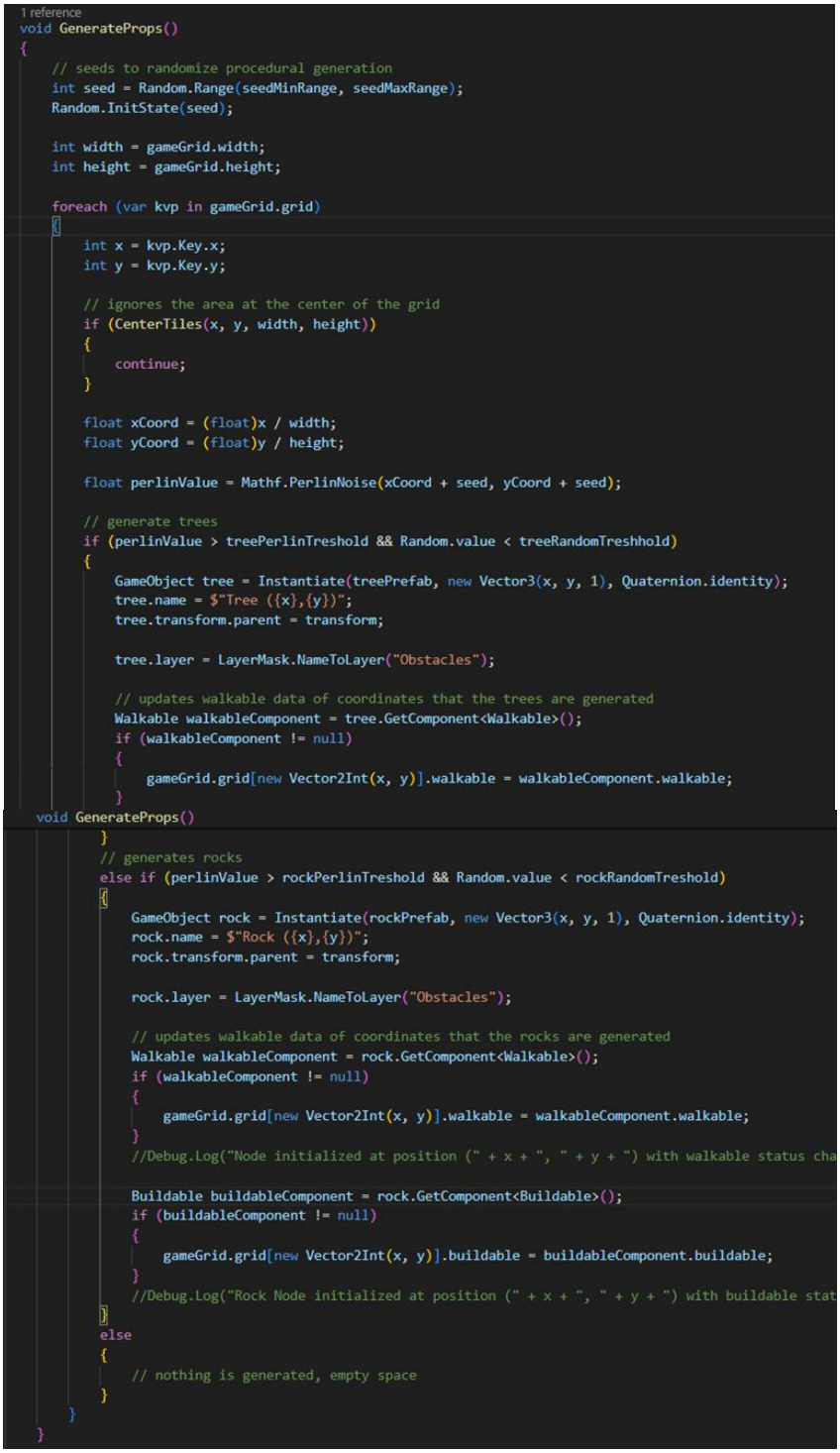


```
1 reference
void PlaceCastleInCenter()
{
    int centerX = width / 2;
    int centerY = height / 2;

    for (int x = centerX - 1; x <= centerX + 1; x++)
    {
        for (int y = centerY - 1; y <= centerY + 1; y++)
        {
            GameObject tilePrefab;

            // Determine if the current tile is a corner or not
            if ((x == centerX - 1 && y == centerY - 1) || (x == centerX + 1 && y == centerY - 1) ||
                (x == centerX - 1 && y == centerY + 1) || (x == centerX + 1 && y == centerY + 1))
            {
                tilePrefab = cornerCastleTilePrefab;
            }
            else
            {
                tilePrefab = centerCastleTilePrefab;
            }

            Vector3 position = new Vector3(x, y, 0);
            GameObject tileGO = Instantiate(tilePrefab, position, Quaternion.identity);
            tileGO.name = "castle tile (" + x + "," + y + ")";

            if (gridParent != null)
            {
                tileGO.transform.SetParent(gridParent);
            }

            Vector2Int gridPosition = new Vector2Int(x, y);
            grid[gridPosition] = new Node(x, y, true, true);  // keep the tiles walkable and buildable
        }
    }
}
```

The Place Castle in Center method in the Game Grid script generates a 3x3 structure on top of the other tiles in the center of the grid. The cookie is in the center of this structure and the node values are ensured to be walkable and buildable.

***Figure 16.***

Generate Props Method

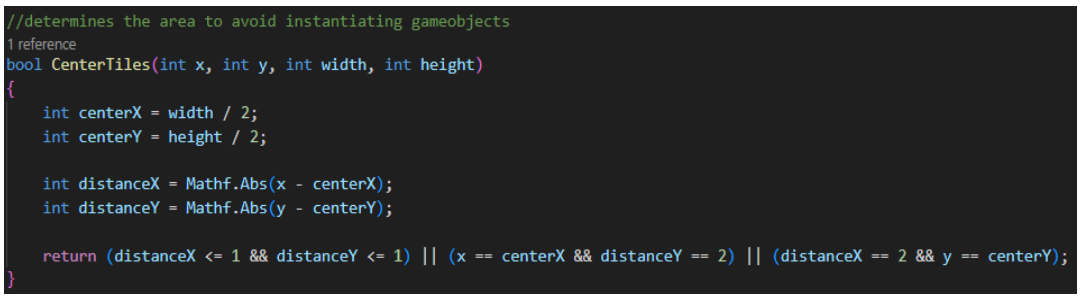


```
1 reference
void GenerateProps()
{
    // seeds to randomize procedural generation
    int seed = Random.Range(seedMinRange, seedMaxRange);
    Random.InitState(seed);

    int width = gameGrid.width;
    int height = gameGrid.height;

    foreach (var kvp in gameGrid.grid)
    {
        int x = kvp.Key.x;
        int y = kvp.Key.y;

        // ignores the area at the center of the grid
        if (CenterTiles(x, y, width, height))
        {
            continue;
        }

        float xCoord = (float)x / width;
        float yCoord = (float)y / height;

        float perlinValue = Mathf.PerlinNoise(xCoord + seed, yCoord + seed);

        // generate trees
        if (perlinValue > treePerlinTreshold && Random.value < treeRandomTreshhold)
        {
            GameObject tree = Instantiate(treePrefab, new Vector3(x, y, 1), Quaternion.identity);
            tree.name = $"Tree ({x},{y})";
            tree.transform.parent = transform;

            tree.layer = LayerMask.NameToLayer("Obstacles");

            // updates walkable data of coordinates that the trees are generated
            Walkable walkableComponent = tree.GetComponent<Walkable>();
            if (walkableComponent != null)
            {
                gameGrid.grid[new Vector2Int(x, y)].walkable = walkableComponent.walkable;
            }
        }
        // generates rocks
        else if (perlinValue > rockPerlinTreshold && Random.value < rockRandomTreshold)
        {
            GameObject rock = Instantiate(rockPrefab, new Vector3(x, y, 1), Quaternion.identity);
            rock.name = $"Rock ({x},{y})";
            rock.transform.parent = transform;

            rock.layer = LayerMask.NameToLayer("Obstacles");

            // updates walkable data of coordinates that the rocks are generated
            Walkable walkableComponent = rock.GetComponent<Walkable>();
            if (walkableComponent != null)
            {
                gameGrid.grid[new Vector2Int(x, y)].walkable = walkableComponent.walkable;
            }
            //Debug.Log("Node initialized at position (" + x + ", " + y + ") with walkable status cha

            Buildable buildableComponent = rock.GetComponent<Buildable>();
            if (buildableComponent != null)
            {
                gameGrid.grid[new Vector2Int(x, y)].buildable = buildableComponent.buildable;
            }
            //Debug.Log("Rock Node initialized at position (" + x + ", " + y + ") with buildable stat
        }
        else
        {
            // nothing is generated, empty space
        }
    }
}
```

In the Generate Props method found in the Perlin Noise script, its role is to procedurally generate environment props based on the Perlin Noise algorithm. It makes use of the seeding system to simulate pseudo randomization in the procedural generation which ensures that every playthrough is unique. The treePerlinThreshold and rockPerlinTreshold determines what parts of the grid would have trees and rocks respectively with the treeRandomTreshold and rockPerlinTreshold randomizes how many trees and rocks are placed on those areas determined by the Perlin thresholds.

**Figure 17.**

Center Tiles Boolean



```
//determines the area to avoid instantiating gameobjects
1 reference
bool CenterTiles(int x, int y, int width, int height)
{
    int centerX = width / 2;
    int centerY = height / 2;

    int distanceX = Mathf.Abs(x - centerX);
    int distanceY = Mathf.Abs(y - centerY);

    return (distanceX <= 1 && distanceY <= 1) || (x == centerX && distanceY == 2) || (distanceX == 2 && y == centerY);
}
```

The Center Tiles Boolean logic determines the area of the center of the grid. It returns the coordinates of the nodes that end up around the center most tile of the grid. This is to ensure that there are no props generated overlapping with other objects in the center area and that the center will not be fully blocked by props.

***Figure 18.***

Try Placing Spawn Point Method

```
1 reference
void TryPlacingSpawnPoint()
{
    Vector2Int spawnPosition = FindValidSpawnPosition();
    if (spawnPosition != Vector2Int.zero)
    {
        spawnPositions.Add(spawnPosition);
        GameObject spawnPointGO = Instantiate(spawnPointPrefab, new Vector3(spawnPosition.x, spawnPosition.y, 0), Quaternion.identity);

        // Set the parent
        GameObject spawnPortalsParent = GameObject.Find("SpawnPortals");
        if (spawnPortalsParent != null)
        {
            spawnPointGO.transform.parent = spawnPortalsParent.transform;
        }
        else
        {
            Debug.LogError("SpawnPortals parent object not found!");
        }

        //Debug.Log("Spawn Point instantiated at position: " + spawnPosition);

        InstantiateTilesForPath(spawnPosition);
    }
    else
    {
        isSpawnPointsPlaced();
        if(isSpawnPointsPlaced())
        {
            Debug.LogWarning("Valid spawn point not found within the grid!");
        }
        if(!isSpawnPointsPlaced())
        {
            Debug.LogWarning("No Valid Spawnpoints have been placed.");
        }
    }
}
```

The Try Placing Spawn Point Method found in the Spawn Points script is responsible for placing the spawn points into the grid.

***Figure 19.***

Instantiate Tiles for Path method

```
1 reference
void InstantiateTilesForPath(Vector2Int spawnPosition)
{
    // Check if a valid path for spawn point
    Pathfinding pathfinding = new Pathfinding(gameGrid);
    List<Node> path = pathfinding.FindPath(spawnPosition, new Vector2Int((int)targetPosition.position.x, (int)targetPosition.position.y));
    if (path != null && path.Count > 0)
    {
        foreach (Node node in path)
        {
            GameObject tileGO = Instantiate(tilePrefab, new Vector3(node.x, node.y, 0), Quaternion.identity);
            GameObject pathParent = GameObject.Find("PathTiles");

            if (pathParent != null)
            {
                tileGO.transform.parent = pathParent.transform;
            }
        }
    }
}
```

The Instantiate Tiles For Path method in the Spawn Points script generates the path that connects the center of the grid and the spawn points generated into the grid.

***Figure 20.***

Code snippet on Find Valid Spawn Position

```
Vector2Int FindValidSpawnPosition()
{
    if (!hasGameGrid)
    {
        return Vector2Int.zero;
    }

    const int maxAttempts = 10000;
    int attempts = 0;

    while (attempts < maxAttempts)
    {
        Vector2Int randomPosition = new Vector2Int(Random.Range(0, gameGrid.width), Random.Range(0, gameGrid.height));

        bool validPosition = true;
        foreach (Vector2Int spawnPos in spawnPositions)
        {
            if (Vector2Int.Distance(randomPosition, spawnPos) < minimumDistanceBetweenSpawnPoints)
            {
                validPosition = false;
                break;
            }
        }

        if (validPosition && IsPathPossible(randomPosition) && gameGrid.grid[randomPosition].walkable && Vector2Int.Distance(randomPosition,
            new Vector2Int((int)targetPosition.position.x, (int)targetPosition.position.y)) >= minimumDistanceFromTarget)
        {
            return randomPosition;
        }
        attempts++;
    }

    return Vector2Int.zero;
}
```

This code snippet for Find Valid Spawn Position is programmed to ensure that the spawn points are placed in locations that are accessible between the potential spawn point location and the center of the grid. The maximum attempt is set to ensure that it is impossible for the method to go into an infinite loop.

***Figure 21.***

Is Path Possible Boolean logic

```
1 reference
bool IsPathPossible(Vector2Int spawnPosition)
{
    if (!hasGameGrid)
    {
        return false;
    }

    Pathfinding pathfinding = new Pathfinding(gameGrid);
    List<Node> path = pathfinding.FindPath(spawnPosition, new Vector2Int((int)targetPosition.position.x, (int)targetPosition.position.y));
    return path != null && path.Count > 0;
}
```

This Is Path Possible boo lean logic is responsible for determining whether or not the path between the spawn points and the center of the grid are connected.

## Operational and Testing Procedure

The application's functionality and execution are concentrated on the operation and testing procedures.

**Functional testing** was performed to ensure that the tower defense game's functionalities functioned as expected. The following procedures are taken to guarantee that each feature worked properly:

1. Verify title screen, main menu, and stage-related functionalities
2. Defined the expected output for each feature.
3. Execute the test cases.
4. Analyze whether the test case results lead to the expected conclusion and determine whether it passes or fails.

**Table 4.**

*Function Suitability Test Case Form (MFSTC-01)*

| **Test Case ID** | MFSTC-01 | **UC Reference** | Title Screen | | |
|---|---|---|---|---|---|
| **Objective** | View and/or Skip Opening Sequence to Title Screen | | | | |
| **Assumptions/ Preconditions:** | The player must be able to see an opening sequence that leads the Title Screen | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Open the application via running the game's executable file | The game starts and the player sees an opening sequence that can be skipped by pressing any key | | | | |
| **Status** | TBD | **Severity** | | **Priority** | |

**Maintainability testing** is performed to guarantee that the game can be adjusted to enhance, fix, or adaptability to environmental or requirement changes.

The following tests should be performed:

1. The game can be set up to run on several different platforms, or it can be used to create additional objects or systems.

2. Analyze the impact of an upcoming modification on the game. This includes diagnosing errors, defects, or the causes of shortcomings.

3. Test whether the game can be modified effectively and efficiently without reducing the quality of either the software or the system or causing new problems.

**Table 5.**

*Maintainability Test Case Form (*MMTC-01)

| **Test Case ID** | MMTC-01 | **UC Reference** | Towers | | |
|---|---|---|---|---|---|
| **Objective** | To assess how easily some gameplay components can be modified or updated. Add a new tower type. | | | | |
| **Assumptions/ Preconditions:** | The player is able to build towers and there are enemies for the towers to target. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Build a new tower. | The player is able to build a new tower in their selected area. | | | | |
| 2. Start Wave | Enemies starts spawning from the portal and moves toward the base. | | | | |
| 3. Enemies get in range of the new tower. | The new tower works targets enemies. The tower's damage, attack speed, and attack range work as intended. | | | | |
| **Status** | TBD | **Severity** | | **Priority** | |

**Evaluation Procedure**

The quality of the developed game system is determined through ISO\IEC 25010 software quality metrics to evaluate the developed program. The respondents consist of 25 students majoring in Information Technology/Computer Science and 6 professionals in the same field, potentially including faculty members and industry professionals. These individuals should possess research experience or knowledge about both AI and Video Games. They assess the system based on relevant criteria. The following evaluation procedure is used as a guide to evaluate the overall quality of the developed system:

1. Invite knowledgeable respondents, including Computer Science/Information Technology students and professionals, to act as evaluators for the system.
2. Distribute a Google Forms link containing a video presentation explaining the system features and demonstrating its processes to online respondents.
3. For face-to-face respondents, conduct a live system demonstration; for online respondents seeking more information, provide an online demonstration.
4. Following the project demonstration, evaluator respondents utilize a form adapted from ISO/IEC 25010 to rate the system based on their experience and observations.
5. Process the completed forms and tabulate the data in Microsoft Excel to calculate mean ratings.
6. Interpret the adjectival ratings for the mean ratings using the Likert Scale presented in the table below.

**Table 6**

*Four-Point Likert Scale*

| Likert Scale | Approval Rating | Rating |
|---|---|---|
| 4 | Highly Acceptable | 3.4-4.0 |
| 3 | Very Acceptable | 2.6-3.3 |
| 2 | Acceptable | 1.8-2.5 |
| 1 | Not Acceptable | 1.0-1.7 |

**Table 7**

*The Range of Weighted Mean Ratings and its Qualitative Representation*

| Likert Scale | Approval Rating | Rating |
|---|---|---|
| 4 | Highly Acceptable | 3.4-4.0 |
| 3 | Very Acceptable | 2.6-3.3 |
| 2 | Acceptable | 1.8-2.5 |
| 1 | Not Acceptable | 1.0-1.7 |

# Chapter 4

# RESULTS AND DISCUSSION

This chapter presents the project description, project structure, project test results, project capabilities and limitations, and project evaluation results of the study.

## Project Description

Slimes Raid Cookie TD is a 2D Pixel Art Tower Defense game that takes inspiration from old school tower defense games and modern procedurally generated games. The game offers varied gameplay challenges from procedurally generated stages that randomize spawn locations, multiple paths, difficulty that scale over time, and a customized algorithm for pathfinding.

Slimes Raid Cookie TD tasks players with strategically positioning defensive structures to fend off waves of enemy units, requiring them to adapt to an evolving map after each wave. This game is exclusively compatible with desktop or laptop computers running Windows 10 or 11. The project is developed using C# and the Unity Game Engine. The artwork consists of custom pixel art created by the team and free online assets.

**Project Structure**

***Figure 22.***

Main Menu Screen


Figure 22 shows the Main Menu Screen upon initializing the game. The player can select the Play button in order to start the game, change the settings of the game through the Options button, check the gallery through the magnifying glass icon, or exit the game through the Exit Game button.

***Figure 23.***

Select Game Mode


Once the player selects the "Play" button on the Main Menu Screen, a modal dialog box will appear on the screen, which is the "Select Game Mode." In the select game mode, it shows three different kinds of waves for the game mode. The Easy game mode ends upon clearing 10 waves of enemies. The Normal game mode ends upon clearing 20 waves of enemies. And then the Survival mode that will only end if the player loses all health points or quits the game.

***Figure 24.***

Main Menu Options Panel


As seen in Figure 24, once the player clicks the Options button from the main menu, the Main Menu Options Panel will pop up. The panel comprises the volume controller for background music and sound effects of the game, the resolution dropdown box for the player to change the screen resolution of the game, and the close button to dismiss the panel.

***Figure 25.***

How to Play Screen


As illustrated in Figure 25, once the player starts the game after selecting a game mode, a “How to Play Screen” will be first displayed on the screen before the game starts moving. This serves as a tutorial of the game to help the users of what the user interface is and the controls.

***Figure 26.***

Procedurally Generated Stage


Figure 26 illustrates the playable stage of the game. The gameplay shows the overall looks of the game such as the user interface housing all the relevant buttons as well as the grid for where the player could interact with.

***Figure 27.***

Life System and Gold System


Figure 28 depicts the heart symbol for life points, representing the chances or the number of lives given to the player. Each time a slime successfully reaches the cookie, the player's life decreases. The players start with a total of 30 lives. The gold system represents the coins or the in-game money that a player can use to purchase a tower to defend it, to upgrade the towers, or to pave way for more room to place towers or manipulate the path of the slimes via removing props such as the trees and the rocks.

***Figure 28.***

Tower Buttons


Figure 28 displays the player's five available towers throughout their gameplay and the delete tool for removing trees and rocks, clicking on the buttons with the tower icons allows the player to place the tower according to their desired location while the red X mark with a tree icon is the delete tool for removing props.

***Figure 29.***

Valid Tower Placement


As shown above in Figure 30, this signifies a Valid Tower Placement with a green highlight color on the tower's model. It can be placed in the tile due to the node being walkable, buildable, and it doesn't completely prevent enemies from getting trapped in an area with no valid paths towards the cookie.

***Figure 30.***

Invalid Tower Placement if path is completely blocked

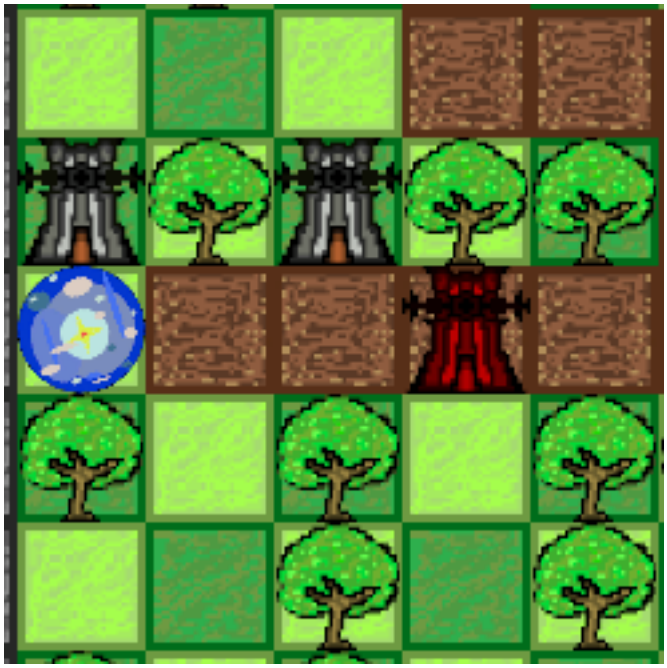


Figure 30 shows a red silhouette on the placed tower showing an invalid tower placement. This is due to one or more of the following, the node or tile that it is going to be placed on is not walkable, not buildable, or if it is placed there then the enemies will not be able to find any valid path towards the target.

***Figure 31.***

Prop Removal and Highlighting of currently hovered Prop


Figure 31 shows the appearance of the lower right tree highlighted in red during the process of being selected to be removed.

***Figure 32.***

Tower Info and Upgrade Panel upon Selection


Figure 32 shows the selected tower's attack range through a visual indicator, the tower's stats on damage and attack interval, upgrade cost, and sell cost upon being removed

***Figure 33.***

Skill Button Off and On Cooldown


In Figure 33 it shows the Freeze skill button, its cost as well its cooldown upon being used. Once used it temporarily stops every enemy that could be found within the map.

***Figure 34.***

Pause/Resume, Fast Forward, Restart, and Menu Buttons


Figure 34 shows the current state of speed of the game and the buttons used to manipulate it. If the game is paused, then the left button shows 2 vertical bars symbolizing pause. If the same button was clicked it will turn into a right-facing triangle. The middle button symbolizes the play speed of the game. If there is only one triangle, then the game plays at normal speed. Clicking the same button turns the button showing 2 triangles instead and changing the speed into fast-forward, which doubles the speed of the game. At its right is the restart button. Clicking it will show a prompt asking you to confirm the restart. While the Bar with Menu written on it is the Menu Button. Clicking it opens the Menu.

***Figure 35.***

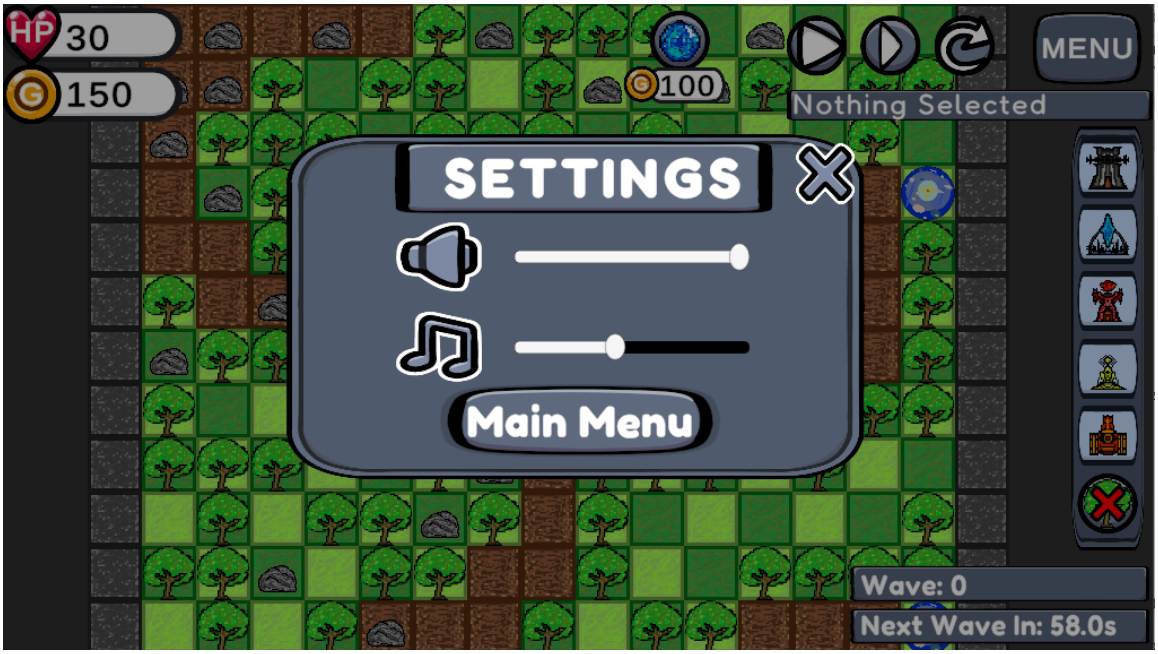
Wave Indicator


Figure 35 illustrates the wave indicator that lets the player know which current wave they are playing. Thus, it displays on the screen the next wave and the timer countdown that includes the time remaining in seconds for the upcoming wave, which informs the players to anticipate the upcoming wave.

.

***Figure 36.***

Menu/Options Setting




Once the player clicks the Menu button, the Options setting panel will pop up, as shown above in Figure 36. The Settings panel contains the volume slider for the game's sound effects, the background music, and the Main Menu button for the game to return to the Main Menu.

***Figure 37.***

Restart Confirmation Prompt


Figure 37 shows the Restart Confirmation prompt after clicking on the restart button between the fast forward button and the menu button on the top right corner of the screen. Clicking on the Checkmark restarts the game while clicking the X Mark resumes the game.

***Figure 38.***

Victory Panel


Figure 39 shows the Victory Panel symbolizing that the player completed the level. There are two buttons the user can click on, the first on the left restarts the game starting a new playthrough in the same game mode while the other returns the screen to the Main Menu.

***Figure 39.***

Defeat Panel

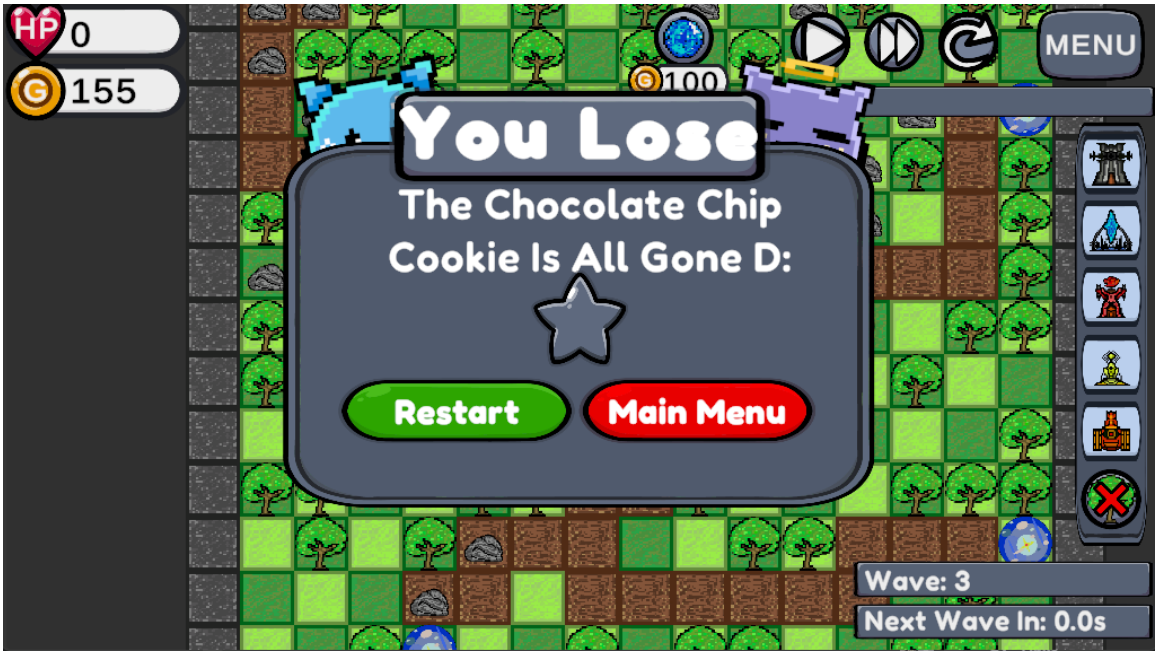
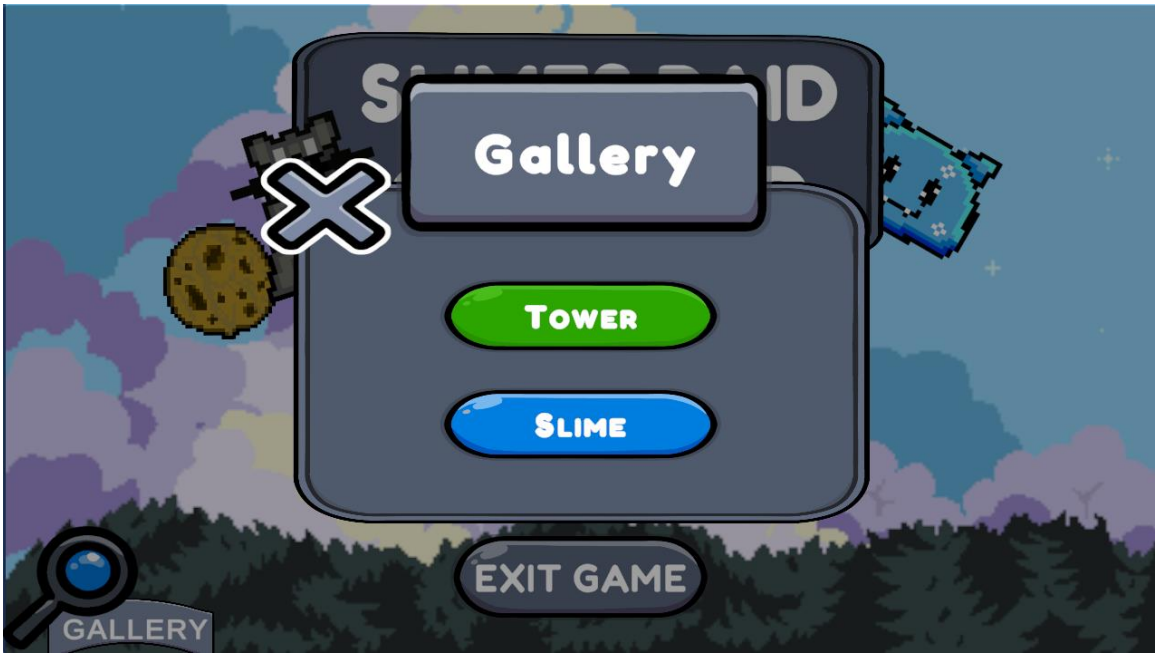



Figure 39 shows the Defeat Panel symbolizing that the player has lost the level and can then select Restart to try again on a new procedurally generated stage on the same game mode or go back to the Main Menu.

***Figure 40.***

Gallery Selection Screen




After the player clicks the Gallery button, as shown above in Figure 40, a panel will pop up on the screen where there is a button for Tower and Slime.

***Figure 41.***

Slime Gallery Screen


When the player chooses the Slime, it will go to the Slime Gallery Screen as shown in Figure 41, which will go to a screen displaying detailed information about every Slime throughout the game. This detailed information refers to the vital statistics of slimes such as Slime's health points, and speed throughout the game. It also shows flavor text or descriptions of the slime monsters.

***Figure 42.***

Tower Gallery Screen


Once the player clicks the Tower button, the player will be directed to the Tower Gallery Screen, as seen in Figure 42. The Tower Gallery Screen provides information about every Tower gameplay.

**Project Test Results**

This section summarized the test results conducted and the data gathered according to the ISO 25010 functional suitability and maintainability.

**Table 8**

*Functional Suitability Test Execution Summary*

| Test Execution | Expected Results | Actual Results | |
|---|---|---|---|
| | | Cycle 1 | Cycle 2 |
| No. of Test Cases Executed | 100% | 100% | 100% |
| Result of Test Cases | | | |
| Passed | 100% | 100% | 100% |
| Failed | 0% | 0% | 0% |
| Not Executed | 0 | 0 | 0 |

The summary of the functional suitability test execution is performed in Table 8. Two cycles were utilized to validate the functional suitability. Both Cycle 1 and Cycle 2 illustrate that every test case was executed successfully. For an accurate list of the executed functional suitability test, please refer to the Appendix D.

**Table 9**

*Maintainability Test Case Form*

| Test Execution | Expected Results | Actual Results | |
|---|---|---|---|
| | | Cycle 1 | Cycle 2 |
| No. of Test Cases Executed | 100% | 100% | 100% |
| Result of Test Cases | | | |
| Passed | 100% | 100% | 100% |
| Failed | 0% | 0% | 0% |
| Not Executed | 0 | 0 | 0 |

The summary of results for the maintainability test case performed is presented in table 9. In each test, case two cycles were performed. Each cycle was performed successfully. For a more accurate list of executed maintainability tests, please refer to Appendix E.

**Project Capabilities and Limitations**

1. Procedural Level Generation: Each playthrough has a unique game session with the in-game level generation randomly generating each playable map area.
2. Tower Placement: Players can place and remove towers on the nodes within the game's grid nodes in order to defend the base from waves of monsters. This can also be used to strategically control the path of the monsters.
3. Tower Variety: There are several towers for the players to choose from, each having its own characteristics in dealing with enemies.
4. Enemy Waves: The game implements an enemy wave system consisting of several waves with different unique enemy types.
5. Customized AI Pathfinding: The AI of the monsters are able to explore and exploit paths resembling that of Ant Colony Systems.
6. Resource Management: Players are given a limited amount of resources in the for of an in-game currency to strategize in defeating waves of enemies and surviving by preventing the enemies from reaching the base.

The following are the limitations of the system:

1. The application is only available on desktop or laptop computers with Windows version 10 or 11.
2. There are no online or multiplayer features in the game.
3. The game may not work on Windows versions that are no longer being supported by Microsoft.

**Project Evaluation**

The evaluation process utilized the ISO 25010 software quality metrics to evaluate the developed program to meet the standard criteria for assessing the quality of the software. The evaluation involved 25 students majoring in Information Technology/Computer Science and 5 professionals in the same field, potentially including faculty members and industry professionals. These individuals should possess research experience or knowledge about both AI and Video Games. They assess the system based on relevant criteria. *The following are the results of the evaluation for 2D Tower Defense.*

**Table 10**

*ISO 25010 Overall Evaluation Results*

| Criteria | Mean Average | Qualitative Interpretation |
|---|---|---|
| Functional Suitability | 3.11 | Very Acceptable |
| Maintainability | 3.15 | Very Acceptable |
| **Overall** | **3.13** | **Very Acceptable** |

Table 10 summarizes the results of the entire ISO 25010 evaluation, including the weighted mean for each criterion and its qualitative interpretation. The Functional Suitability criterion yielded a weighted mean of 3.11, which was evaluated as Very Acceptable. The Maintainability criterion yielded a weighted mean of 3.15, assessed as

Very Acceptable. Overall, the weighted mean of the two criteria was 3.14, which implies that it is very acceptable.

***Figure 43.***

Functional Suitability Testing Result

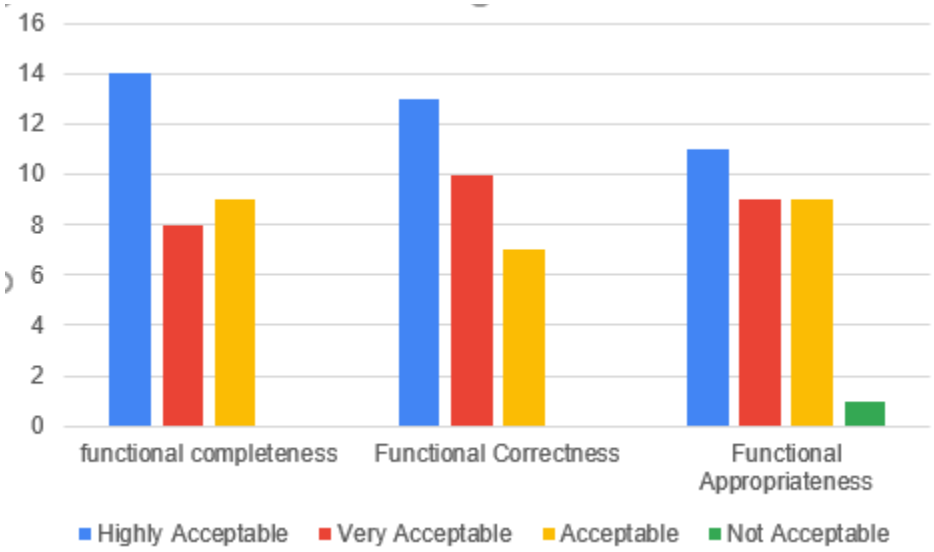



Figure 43 represents the functional suitability testing results graph containing Functional Completeness with a weighted mean of 3.16, Functional Correctness with a weighted mean of 3.16, and Functional Appropriateness with a weighted mean of 3. These scores signify that the players were satisfied with the game's functionalities.

***Figure 44.***

Maintainability Testing Result

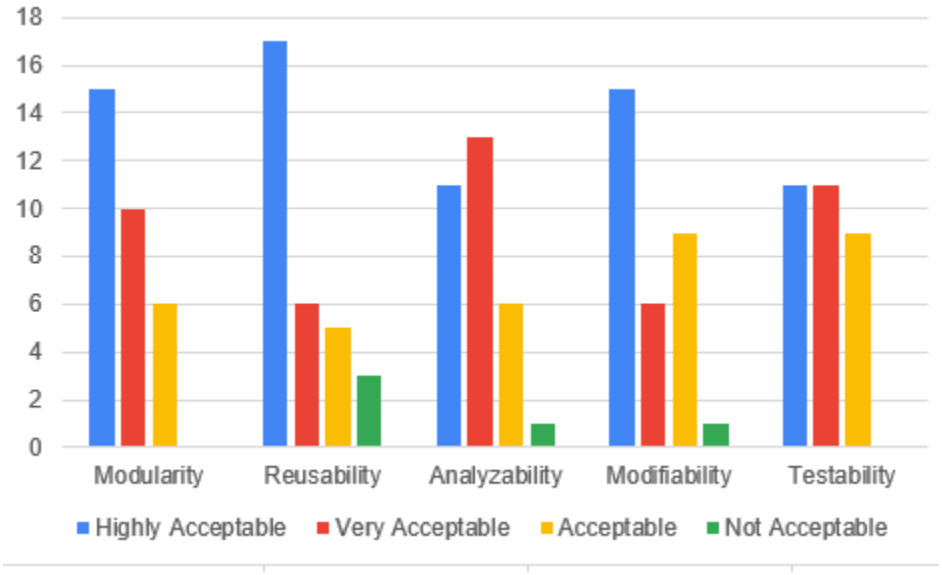



Figure 44 presents the graph for the Maintainability Testing results, which includes Modularity with a weighted mean of 3.29, Reusability with a weighted mean of 3.19, Analyzability with a weighted mean of 3.10, Modifiability with a weighted mean of 3.13, and Testability with a weighted mean of 3.06. These scores signify that the players were satisfied with the maintainability of the game.

# Chapter 5

# SUMMARY OF FINDINGS, CONCLUSIONS AND RECOMMENDATIONS

## Summary of Findings

In this chapter, titled "Slimes Raid Cookie TD," the tower defense game developed as part of this thesis project underwent a comprehensive evaluation based on the ISO/IEC 25010 standard focused on the Functional Suitability and Maintainability of Software. The evaluation process aimed at gauging the effectiveness of the game by meeting its functional requirements and its potential for maintenance and future development. This section of the paper presents the findings of the evaluation, including a comprehensive analysis of the scores obtained from each criterion, derived conclusions based on the results, and the recommendations of the respondents for future improvements of the project. Through analyzing the collected data during the evaluation phase, the study has achieved the following insights:

- Following the standards outlined in ISO 25010, the tower defense game underwent a thorough evaluation on its Functional Suitability resulting in a "very acceptable" rating with an average score of 3.11. This rating signifies that the game effectively aligns with the requirements and expectations of its users.
- Based on the evaluation on maintainability, the results received by this thesis study is a "very acceptable" score rating of an average of 3.15, suggesting that the codebase and architecture of the game are structured in a manner that facilitates easy maintenance and ease in modifying for future development is in a very acceptable extent.

**Conclusions:**

In conclusion, the development of Slimes Raid Cookie TD, a tower defense game with procedurally generated stages and dynamic enemy AI pathfinding has achieved its objectives. Through a comprehensive design and implementation process, incorporating the following key components of the game:

1. The procedural content generation system incorporated into the grid generation and randomized object placement using Perlin Noise enhances the replayability and provides more opportunity for players to create a unique gameplay experience per playthrough.
2. The implementation of Ant Colony Optimization on the pathfinding of enemy monster AI introduces a dynamic level of randomness and adaptive challenges, enriching gameplay experience and promoting strategic decision making among players.
3. The inclusion of systems handling health points and in-game currency adds depth to the gameplay, requiring players to make strategic decisions on resource management.
4. With the various types of slime monsters and different tower types encourage players to experiment with strategies in determining the best strategy, fostering an engaging and adaptable gameplay experience.
5. Slimes Raid Cookie TD received an overall weighted mean of 3.13 that can be interpreted as a “Very Acceptable” rating based on the criteria of ISO 25010 on Functional Suitability and Maintainability.

**Recommendations:**

Based on the findings and conclusions of the study as well as the recommendations of the respondents, the following suggestions are provided for potentially enhancing the game in the future:

1. Expand the procedural generation system of the game by allowing the players to manually select various parameters of the procedural generation system such as inputting the seed to replay on a specific map layout, changing the map size into a larger or smaller size, and other aspects of procedural generation.
2. Consider adjusting the pacing and difficulty of the game to a wider range of players. Some players may find the game's current pacing to be too fast or the difficulty spike being too early making it difficult to clear, potentially limiting their ability to clear and enjoy the game. Therefore, implementing options to adjust game difficulty through settings or part of the selection of game mode with a sufficient tutorial and guidance feature to enhance the accessibility and accommodate players with varying skill levels and preferences.
3. Introduce more variety of mechanics, enemies, and towers to the game to enrich and prolong player experience and engagement. By expanding the game's content with more enemy types, unique tower abilities, and gameplay mechanics players will have more ways to solve problems strategically while also increasing replayability by ensuring a compelling and dynamic experience for new and veteran players alike.
4. Adjust the color palette of the sprites used to be easier to see and distinguish from the surrounding objects.

# REFERENCES


Alexis KB Sounds. (2022, November 16). Ice Crack Freeze Sound Effect. [Video]. YouTube. https://www.youtube.com/watch?v=Xt-vprYxIm4

Andre, T. (2022). *Adaptive pathfinding and utility AI implementation for a tower defense game. Final Year Project (FYP)*, Nanyang Technological University, Singapore. https://hdl.handle.net/10356/156729

Ashlock, D. (2010). *Automatic generation of game elements via evolution.* Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG2010, 289–296. https://doi.org/10.1109/ITW.2010.5593341

Asoftclick. (2022, October 31). Graphicsgale vs aseprite: Which is the best pixel art program? https://asoftclick.com/graphicsgale-vs-aseprite/

Avery, P., Togelius, J., Alistar, E., & Leeuwen, R. (2011). *Computational Intelligence and Tower Defence Games*. 2011 IEEE Congress of Evolutionary Computation, CEC 2011, 1084–1091. https://doi.org/10.1109/CEC.2011.5949738

Andre, T. (2022). *Adaptive pathfinding and utility AI implementation for a tower defense game. Final Year Project (FYP)*, Nanyang Technological University, Singapore. https://hdl.handle.net/10356/156729

Becker, A. & Görlich, D. (2020). What is Game Balancing? An Examination of Concepts. 10.55969/paradigmplus.v1n1a2.

Boutros, D. (2023, November 9). Difficulty is difficult: Designing for hard modes in games. Game Developer. https://www.gamedeveloper.com/design/difficulty-is-difficult-designing-for-hard-modes-in-games

Coding Quest. (2023, November 16). Procedural World Generation with perlin noise. YouTube. https://www.youtube.com/watch?v=eiM_AStmeGE&t=12s

Connor, A., Greig, T., & Kruse, J. (2017). *Evaluating the Impact of Procedurally Generated Content on Game Immersion.* The Computer Games Journal, 6, 1–17. https://doi.org/10.1007/s40869-017-0043-6

Dias, D. M. P., SOUSA, J. P. P., BARROSO, B. C., & MAGALHAES, I. M. (2022, September). A Novel Procedural Content Generation Algorithm for Tower Defense Games. In *Proceedings of the 6th International Conference on Algorithms, Computing and Systems* (pp. 1-7). https://doi.org/10.1145/3564982.3564993

Dorigo, M., & Stützle, T. (2004). MIT Press. Retrieved November 30, 2023, from https://web2.qatar.cmu.edu/~gdicaro/15382/additional/aco-book.pdf

DOVA-SYNDROME Youtube Official. (2017, November 6). クレイジークッキング @ フリーBGM DOVA-SYNDROME OFFICIAL YouTube CHANNEL. [Video]. YouTube. https://www.youtube.com/watch?v=asfDxQQZEnU

DOVA-SYNDROME Youtube Official. (2020, August 4). 明日も晴れるといいね @ フリーBGM DOVA-SYNDROME OFFICIAL YouTube CHANNEL. [Video]. YouTube. https://www.youtube.com/watch?v=kgNXP6FP3RY

Green, D. (2016). Procedural content generation for C++ game development: Get to know techniques and approaches to procedurally generate game content in C++ using simple and Fast Multimedia Library. Packt Pub.

Keith6535. (2022, February 25). Free 8-bit mystery theme music for video game development [Video]. YouTube. https://www.youtube.com/watch?v=LDPFCCkfdYE

Koishi Komeiji The YouTuber 2023. (2022, August 4). PSX - Press Button Sound Effects (Download Sound). [Video]. YouTube. https://www.youtube.com/watch?v=9wiI5iio-fc

*Krita FAQ*. (n.d.). https://docs.krita.org/en/KritaFAQ.html#what-is-krita

Lankoski, P., & Holopainen, J. (2017). *Game Design Research: An Overview*. Retrieved from https://www.researchgate.net/publication/323748642_Game_Design_Research_An_Overview

Loguidice, B. (2009, May 5). *The history of rogue: Have @ you, you deadly zs. Game Developer.* https://www.gamedeveloper.com/design/the-history-of-rogue-have-you-you-deadly-zs

Martek, C. (2012). Procedural generation of road networks for large virtual environments. Rochester Institute of Technology.

MasterClass. (2021, July 19). *Tower Defense Game Genre: 6 Characteristics of TD Games - 2023* - MasterClass. https://www.masterclass.com/articles/tower-defense-game-video-game-guide

McCooey, C. (2021). *PROCEDURAL GENERATION OF GAME-WORLDS* (Open Access Master's Thesis No. 2033). University of Rhode Island. Retrieved from https://digitalcommons.uri.edu/theses/2033

Mount, D. (2019). CMSC 425: Lecture 14 *Procedural Generation: Perlin Noise*. Retrieved November 30, 2023, from https://www.cs.umd.edu/class/fall2019/cmsc425/handouts/lect14-perlin.pdf

Musgrave, F. K., Peachey, D., Perlin, K., & Worley, S. (2003). *Texturing and modeling: A procedural approach* (D. S. Ebert, Ed.). Morgan Kaufmann.

Nakov, S., & Kolev, V. (2013). *Fundamentals of Computer Programming with C#*: The Bulgarian C# Book. Faber Publishing.

Nelon, H.. (n.d.). *Esports Explained: MOBA.* KemperLesnic. Retrieved from https://www.kemperlesnik.com/2020/03/esports-explained-mobas/

Neumann, F., & Witt, C. (2006). *Ant colony optimization algorithm*. Secretary of the SFB 531.

Öhman, J. (2020). *Procedural Generation of Tower Defense Levels* (Dissertation). Retrieved from https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-166594

Ommzi Admin. (2018, April 24). Importance of user interface and user experience in gaming. Ommzi Solutions - Game Development Company. https://www.ommzi.com/importance-user-interface-user-experience-gaming/

Peek, S. (2023, February 21). Photoshop: The uses, benefits and alternatives. business.com. https://www.business.com/articles/photoshop-reigns-supreme-how-the-software-has-maintained-market-dominance/

Rabin, S. (2020). Game AI Pro 360: Guide to Movement and Pathfinding. CRC Press/Taylor & Francis Group.

Rouse, M. (2015, June 19). Pixel Art. Techopedia. https://www.techopedia.com/definition/8884/pixel-art

Sahuri, G., Rosalina, R., & Welly Tulili Panandu, H. (2023). *Tower Defense Game based on 2D Grid Using Goal-Based Pathfinding Method. International Journal of Management Science and Information Technology*, 3(1), 1–7. https://doi.org/10.35870/ijmsit.v3i1.819

Sandra Michelle. (2019, June 21). Electricity Sound Effect xd (Thx for 1M Views!). [Video]. YouTube. https://www.youtube.com/watch?v=TEFn1sB_XzI

Schreiber, I. (2009, August 19). Level 16: Game balance. Game Design Concepts. https://gamedesignconcepts.wordpress.com/2009/08/20/level-16-game-balance/

Schreiber, I., & Romero, B. (2022). Game balance. CRC Press, Taylor & Francis Group.

Sergeev, A. (2017, June 26). *Graphicsgale - a free tool for spriting and Pixel Art*. 80lv. https://80.lv/articles/graphicsgale-a-free-tool-for-spriting-and-pixel-art/

SoundMaster. (2012, August 15). Ice Sound Effects. [Video]. YouTube. https://www.youtube.com/watch?v=JCu7q5gZaeA

SoundX. (2021, July 16). Laser Beam Cannon - Sound Effect | Free Sound Effect. [Video]. YouTube. https://www.youtube.com/watch?v=DbQG8R918hI

Sidhu, H. K. (2020). *Performance Evaluation of Pathfinding Algorithms* (Electronic Theses and Dissertations No. 8178). University of Windsor. Retrieved from https://scholar.uwindsor.ca/etd/8178

Sticky C. (2023, April 26). *The entire History of Tower Defense games, i guess* [Video]. YouTube. https://www.youtube.com/watch?v=7KRszo2vZ5Y

Tank, S. (2023). *ARTIFICIAL INTELLIGENCE IN GAMES AND ITS APPLICATION*. Retrieved from https://www.researchgate.net/publication/366907716_ARTIFICIAL_INTELLIGENCE_IN_GAMES_AND_ITS_APPLICATION

Tokio. (2023, February 13). What is Pixel Art? Retrieved from https://www.tokioschool.com/en/news/what-is-pixel-art/

Wirtz, B. (2023, July 17). Game balance: The odds are not always in your favor. Video Game Design and Development. https://www.gamedesigning.org/learn/balance/

Wolf, M. J. P. (2008). The video game explosion: A history from pong to PlayStation and beyond. Greenwood Press.

Visual Studio Code. (2021). *Documentation for Visual Studio code*. Retrieved December 3, 2023, from https://code.visualstudio.com/docs

Visual Studio Code (2021). *Visual Studio Code Frequently asked questions*. Retrieved December 3, 2023 from. https://code.visualstudio.com/docs/supporting/FAQ

ZapSplat Sound Effects. (2024, February 15). Fireball Explosion Sound Effect. [Video]. YouTube. https://www.youtube.com/watch?v=EkAixTSqq08

Zenva GameDev Academy. (n.d.). *What is Unity?* Retrieved from https://gamedevacademy.org/what-is-unity/

2MirrorsDialogue. (2017, November 28). Stone Hit Sound Effect. [Video]. YouTube. https://www.youtube.com/watch?v=MGYtG3I_sIY

# Appendix A

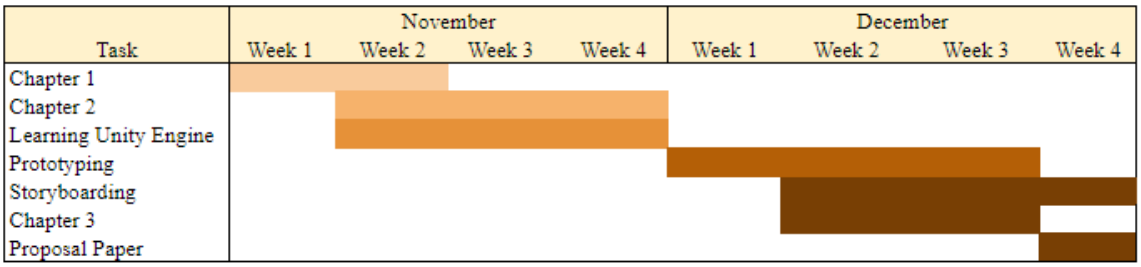
## Budgetary Requirements

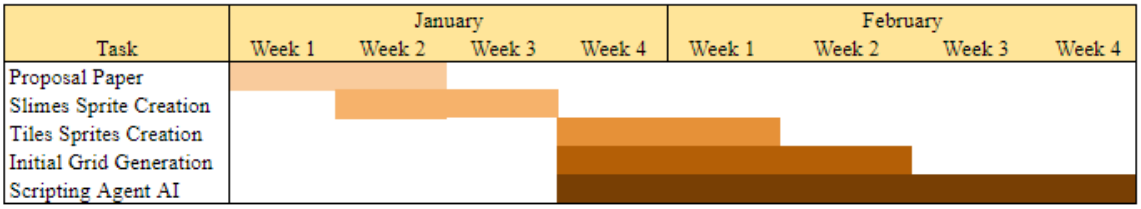
In the development of Slimes Raid Cookie TD, the proponents of the thesis project used the following software and development tools:

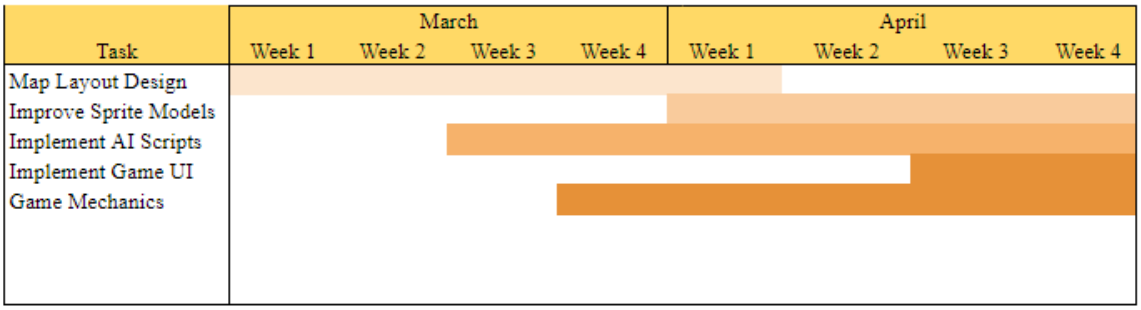
| Item | Purpose | Budget |
|---|---|---|
| Unity Game Engine (Personal License) | The game engine used for developing and implementing the components of the Tower Defense game | Free |
| GraphicsGale | A light weight portable software used in creating digital pixel art. It was used for creating the sprites for the environment, enemies, towers, and the projectiles. | Free |
| Krita | A free and open source painting program used for creating and modifying the sprites used within the game. | Free |
| Visual Studio Code | A source code editing software used for developing and implementing the essential codes or scripts used in the game. | Free |
| Facebook Messenger | This is a mobile communication tool for discussions between group members over a long distance. | Free |
| Discord | An alternative communication tool to Facebook Messenger for its accessibility features that allow easy | Free |
| Google Docs | Collaborative Online Word Processor tool used by the proponents of the study to easily create, modify, and distribute writing of the manuscript. | Free |
| Google Drive | The Cloud Storage system that is used in the thesis project for easy transferring of project files between the researchers. | Free |

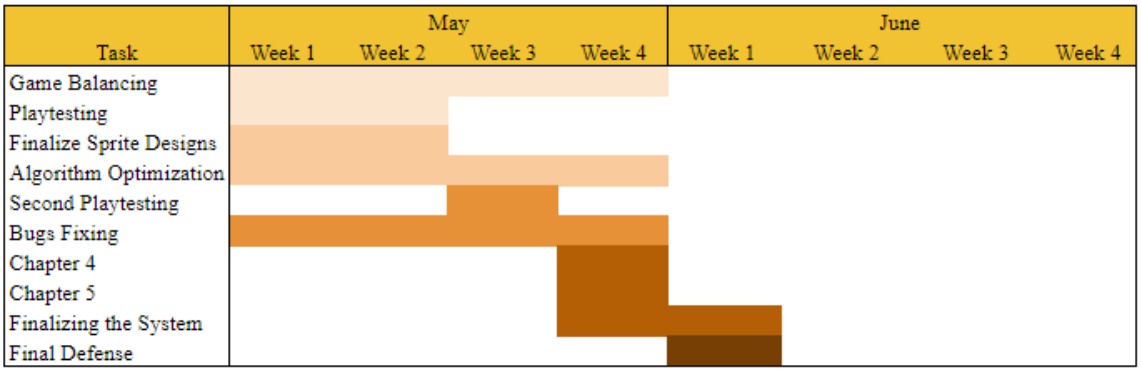
# Appendix B

## Gantt Chart



## Appendix C

## Software Evaluation Instrument of ISO 25010 2D Tower Defense

**Name: (Optional)_____**

**Occupation (Optional): _____**

**Age: _____**

Instruction: Please Evaluate the software material using the given scale and placing a checkmark (✓) under the corresponding numerical rating:

Numerical Equivalent

**4 –** Highly Acceptable **3 –** Very Acceptable

**2 –** Acceptable **1 –** Not Acceptable

| **A. Functional Suitability** | | | | | |
|---|---|---|---|---|---|
| **Indicators** | | **4** | **3** | **2** | **1** |
| Functional Completeness | The set of Functions cover all the specified tasks and user objectives | | | | |
| Functional Correctness | The software provides the correct results with the needed degree of precision | | | | |
| Functional Appropriateness | The functions facilitate the accomplishment of specified tasks and objectives | | | | |
| **B. Maintainability** | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Modularity | The software is made of separate components so changes to one have minimal impact on others. | | | | |
| Reusability | The product can be used in multiple systems or for building other products. | | | | |
| Analysability | The ease of assessing the impact of changes, diagnosing issues, and identifying parts for modification. | | | | |
| Modifiability | The software can be modified efficiently without introducing defects or reducing quality. | | | | |
| Testability | The ease and efficiency of setting test criteria and performing tests to meet those criteria. | | | | |

# Appendix D

## Functional Test Cases

| **Test Case ID** | MFSTC-01 | **UC Reference** | Title Screen | | |
|---|---|---|---|---|---|
| **Objective** | View and/or Skip Opening Sequence to Title Screen | | | | |
| **Assumptions/ Preconditions:** | The player must be able to see an opening sequence that leads the the Title Screen | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Open the application via running the game's executable file | The game starts and the player sees an opening sequence that can be skipped by pressing any key | | | | |
| **Status** | PASSED | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-02 | **UC Reference** | Main Menu Screen | | |
|---|---|---|---|---|---|
| **Objective** | Access the Main Menu Screen and navigate through the menus | | | | |
| **Assumptions/ Preconditions:** | The player must be able to interact with the buttons present in this menu. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Wait for the opening sequence to finish or skip.<br>2. Loads the Main Menu<br>3. Interact with the buttons present in the scene.<br>4. Press ESC to go back from one screen to the previous | The opening sequence ends and shows the main menu screen that has a Play button, Settings button, Gallery button, and Exit button.<br><br>Play button shows a game mode menu. Settings opens a settings panel that has volume and screen resolution. Gallery opens a selection screen that leads to the slime and tower gallery.<br><br>Pressing ESC will close active panels and going back to the main menu screen. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-03 | **UC Reference** | Game Mode Select | | |
|---|---|---|---|---|---|
| **Objective** | View the game mode selection screen and then select any of the available modes. Determine that the correct game mode is implemented. | | | | |
| **Assumptions/ Preconditions:** | The player must be able to interact with the buttons in the game mode selection screen. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click the Play button in the Main Menu.<br>2. Click on any of the available options.<br>3. Test if the game mode is properly implemented | The game is able to run the specified number of waves according to the game mode. Player wins upon reaching wave 10 or 20 on the respective game modes. Player is able to go on for longer than 20 waves in endless mode. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-04 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Place towers on the grid. | | | | |
| **Assumptions/ Preconditions:** | The player must be able to place towers on the nodes on the grid. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click on any tower button on the top part of the screen.<br>2. Hover on tiles on the grid<br>3. Place the tower on a valid tile. | The player is able to place a tower on valid tiles and the player will not be able to place towers on invalid conditions such as on top of trees. rocks or other towers. Should be unable to place towers with insufficient gold amount. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-05 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Remove Trees and Rocks on the grid | | | | |
| **Assumptions/ Preconditions:** | The player must be able to remove trees or rocks on the grid while having sufficient gold. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click on the remove prop button.<br>2. Hover on any selected tree or rock.<br>3. Click on the selected tree or rock to remove | The player is able to remove obstacles or props on the grid after clicking and then only losing gold if a tree or rock is removed. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-05 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Selecting, Upgrade, and Selling Towers | | | | |
| **Assumptions/ Preconditions:** | The player has placed towers on the grid and is able to select the towers. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Select a tower on the grid.<br>2. Click on Upgrade Button<br>3. Click on Sell Button | The player should be able to select a tower. The tower selected would show a transparent circle signifying its attack radius and a highlight tile, The player is able to upgrade the tower and the amount of gold deducted matches the value shown on the tower's status panel. The player is able to sell the tower and receive gold upon selling the tower. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-06 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Selecting, Upgrade, and Selling Towers | | | | |
| **Assumptions/ Preconditions:** | The player has placed towers on the grid and is able to select the towers. | | | | |
| **Actions** | **Expected Results** | | | | |
| 4. Select a tower on the grid.<br>5. Click on Upgrade Button<br>6. Click on Sell Button | The player should be able to select a tower. The tower selected would show a transparent circle signifying its attack radius and a highlight tile, The player is able to upgrade the tower and the amount of gold deducted matches the value shown on the tower's status panel. The player is able to sell the tower and receive gold upon selling the tower. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-07 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Freeze All Enemies | | | | |
| **Assumptions/ Preconditions:** | The player has at least 100 gold and there are enemies present on the scene | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click on the Freeze Skill Button | The button plays a freezing sound effect, freezes all enemies (indicated by a change in color into blue), and all the enemies are prevented from moving further. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-08 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Game Speed Control | | | | |
| **Assumptions/ Preconditions:** | The player is able to view and interact with the buttons on the upper right corner of the scene. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click on the play button.<br>2. Click the fast forward button. | If the game is not paused, pause the game. Else resumes the game.<br><br>If the game is not paused, it speeds up the game. Else resumes the game. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-09 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Restart the current game mode. | | | | |
| **Assumptions/ Preconditions:** | The player is able to view and interact with the restart button on the upper right corner of the scene. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click on the restart button<br>2. Click the checkmark button. | Click the restart button to see a confirmation prompt.<br><br>Click the checkmark button and the scene restarts generating all objects on the game scene. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-10 | **UC Reference** | In-Game Screen | | |
|---|---|---|---|---|---|
| **Objective** | Open the Main Menu Screen and Interact with the volume settings then get back to the main menu. | | | | |
| **Assumptions/ Preconditions:** | The player is able to view and interact with the menu button on the upper right corner of the scene. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click the menu button<br>2. Change volume by moving the slider or clicking the mute buttons.<br>3. Click the Main Menu button | A menu panel of the options for the sound effects and music volume after clicking the menu button. Adjusting the volume sliders changes the volume. Clicking on the mute buttons mute and unmutes the sounds. Clicking the Main Menu button sends the user back to the Main Menu | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-11 | **UC Reference** | Gallery Screen | | |
|---|---|---|---|---|---|
| **Objective** | Navigate the Gallery and see if the user can move back and forth from the main menu screen by pressing the close buttons or pressing ESC key | | | | |
| **Assumptions/ Preconditions:** | The player is able to view and interact with the magnifying glass icon on the bottom left of the Main Menu screen. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click gallery selection menu.<br>2. Click Slimes button<br>3. Click on the slime icons.<br>4. Go Back a Screen<br>5. Click on the Towers button<br>6. Click on the tower icons | A menu panel of the gallery shows up. Clicking on the slimes button leads to the slimes gallery where the player can view details about the slimes by clicking on the icons. Pressing the ESC key or the X button on the screen should go back to the previous panel. Clicking on the towers button leads to the tower gallery. Pressing the tower icons show details about the selected tower. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MFSTC-12 | **UC Reference** | Settings Screen | | |
|---|---|---|---|---|---|
| **Objective** | Navigate the Settings, adjust the volume, and change screen resolution. | | | | |
| **Assumptions/ Preconditions:** | The player is able to view and interact with the settings button under the play button. The player doesn't have speakers or headphones muted. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Click the options button from the main menu.<br>2. Adjust Volume Sliders<br>3. Click on the mute buttons.<br>4. Change screen resolution | An options panel is shown to the user. Adjusting the panels correctly changes the volume of sound effects and music. Clicking on the mute button silences respective assigned sound effects or music, then unmuting should let the game return volume to previous settings | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

# Appendix E

## Maintainability Test Cases

| **Test Case ID** | MMTC-01 | **UC Reference** | Towers | | |
|---|---|---|---|---|---|
| **Objective** | To assess how easily some gameplay components can be modified or updated. Add a new tower type. | | | | |
| **Assumptions/ Preconditions:** | The player is able to build towers and there are enemies for the towers to target. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Build a new tower..<br><br>2. Start Wave<br><br>3. Enemies get in range of the new tower. | The player is able to build a new tower in their selected area.<br><br>Enemies start spawning from the portal and move toward the base.<br><br>The tower targets the enemies. Deals as much damage as its specified, attacks on the specific attack interval, and appropriate attack range based on circle collider size. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MMTC-02 | **UC Reference** | Slime Enemies | | |
|---|---|---|---|---|---|
| **Objective** | To assess how easily some gameplay components can be modified or updated. Add a new slime type. | | | | |
| **Assumptions/ Preconditions:** | The player is able to see that the new slime spawns and is able to deal with it accordingly by placing towers. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Slime spawns on the next wave<br>2. Slime gets hit.<br>3. Slime gets destroyed. | The new slime enemy starts spawning from the portal and moves toward the base. It is being targeted and takes damage. It is able to move similarly as the other types of slime previously created. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

| **Test Case ID** | MMTC-02 | **UC Reference** | Slime Behavior | | |
|---|---|---|---|---|---|
| **Objective** | Modify parameters from a slime and test change in behavior. | | | | |
| **Assumptions/ Preconditions:** | The slime is able to path from the spawn point towards the target destination. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Slime spawns on the next wave<br>2. Slime moves on the grid, | The slime enemy starts spawning from the portal and attempts to move forward to the base. It may randomly select the next node based on changed parameters or not. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

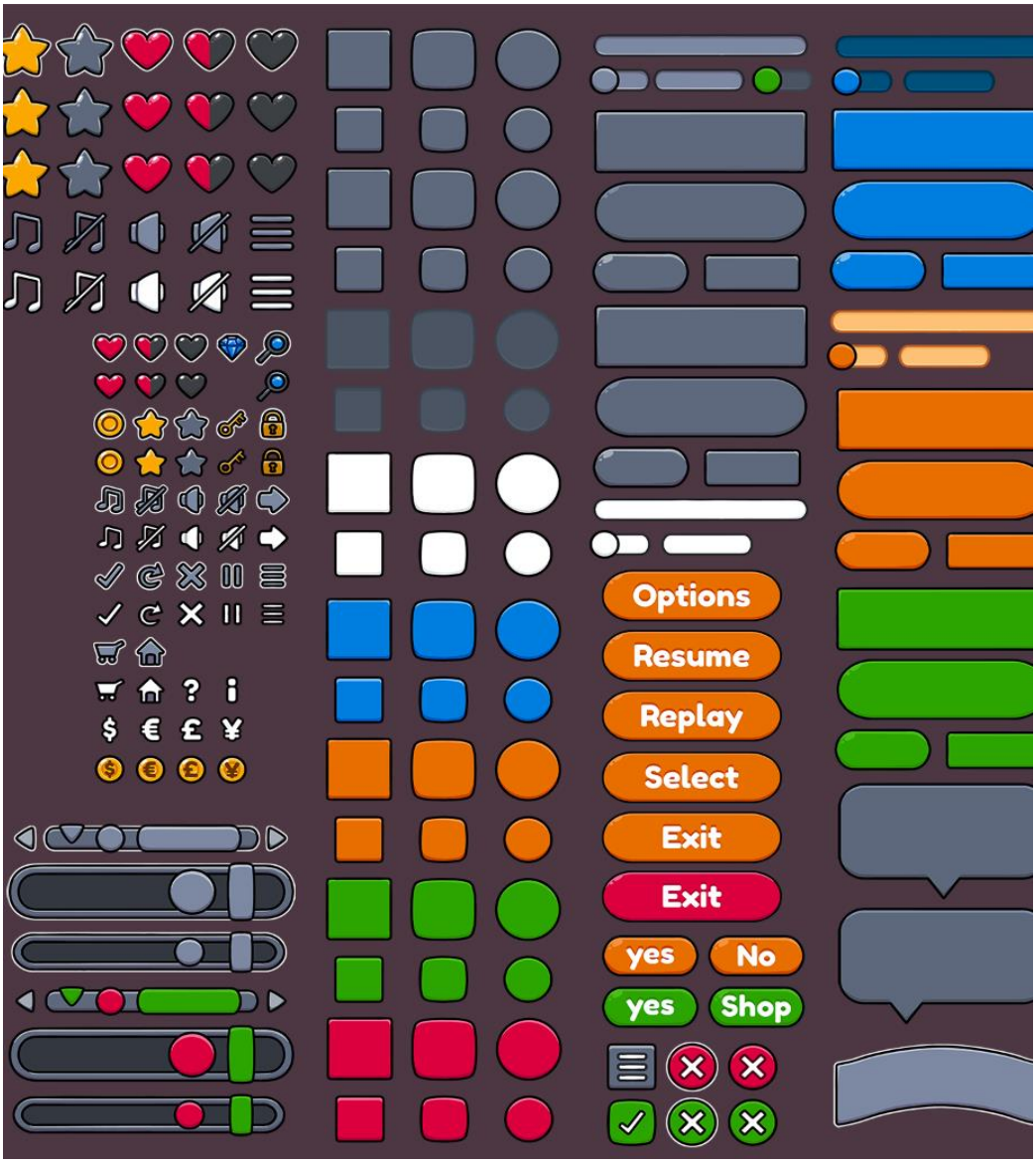
| **Test Case ID** | MMTC-03 | **UC Reference** | Wave Spawning System | | |
|---|---|---|---|---|---|
| **Objective** | Verify that the slimes are being created and destroyed as intended based on the spawn system logic so that the waves will start after the previous wave ends. | | | | |
| **Assumptions/ Preconditions:** | The slime is able to path from the spawn point towards the target destination. | | | | |
| **Actions** | **Expected Results** | | | | |
| 1. Slime spawns on the next wave<br>2. Slime moves on the grid either reaches the target or gets destroyed. | The system is able to check conditions properly then the next wave starts if all slimes of the wave are destroyed either by reaching the target destination or defeated by taking damage from towers. | | | | |
| **Status** | Passed | **Severity** | | **Priority** | |

# Appendix F

## Sprite Sheet

# Appendix G

Penzilla's Basic GUI Bundle Sheet



# Appendix H

## samvieten's Pixel Art Painting of Clouds.

# Appendix I

## Certificate of Similarity Index Using Turnitin URDS

| | | | |
|---|---|---|---|
| | **TECHNOLOGICAL UNIVERSITY OF THE PHILIPPINES**<br>Ayala Blvd., Ermita, Manila, 1000, Philippines<br>Tel No. +632-5301-3001 local 711 \| Fax No. +632-521-4063<br>Email: urds@tup.edu.ph \| Website: www.tup.edu.ph | Index No. | REF-URD-INT-CSI |
| | | Issue No. | 01 |
| | | Revision No. | 01 |
| | | Date | 04132021 |
| VRE-URD | **CERTIFICATE OF SIMILARITY INDEX USING TURNITIN** | Page | 2 / 5 |
| | | QAC No. | CC-04132021 |

This is to certify that the manuscript entitled

**"DEVELOPMENT OF A TOWER-DEFENSE GAME WITH PROCEDURALLY GENERATED STAGES"**

authored by

**DAN EIAIEL F. EBOÑA**
**MAURICE H. LAGAS**
**CYRENE ROSE C. MANIBALE**
**KEITH C. ROSETE**

Has been subjected to similarity check on June 13, 2024 using Turnitin with generated similarity index of 9%.

Processed by:

**DOROTHY P. PERNIS**
Staff, URDS

Certified correct by:

**FRANCISCO D. ESPONILLA II, LPT, Ed.D.**
Director, URDS

| | |
|---|---|
| Transaction ID | |
| Signature | |

# Appendix J

## Thesis Grammarian Certification

| | | | |
|---|---|---|---|
| | **TECHNOLOGICAL UNIVERSITY OF THE PHILIPPINES**<br>Ayala Blvd., Ermita, Manila, 1000, Philippines<br>Tel No. +632-5301-3001 local 608| Fax No. +632-8521-4063<br>Email: cos@tup.edu.ph | Index No. | |
| | | Revision No. | |
| | | Effectivity Date | |
| VAA-COS | **THESIS GRAMMARIAN CERTIFICATION** | Page | |

This is to certify that the thesis entitled,

DEVELOPMENT OF A TOWER-DEFENSE GAME WITH PROCEDURALLY GENERATED STAGES

authored by

Dan Eiaiel F. Eboña
Maurice H. Lagas
Cyrene Rose C. Manibale
Keith C. Rosete

has undergone editing and proofreading by the undersigned.

This Certification is being issued upon the request of Dan Eiaiel F. Eboña, Maurice H. Lagas, Cyrene Rose C. Manibale, And Keith C. Rosete for whatever purposes it may serve them.

**MS. FRANZE NAVARRO OROCEO**
Grammarian

Technological University of the Philippines

June 12, 2024

| | |
|---|---|
| Transaction ID | |
| Signature | |

# Researchers Profile

## KEITH C. ROSETE

**Email:** rosetekeith@gmail.com || **Mobile No.:** 0960-904-9515

**Address:** 385 Alabang-Zapote Rd., Tatang Exequiel Compound., Unit C5 Dela Cruz Apartment, Talon Uno, Las Piñas City

### CAREER OBJECTIVE

Graduating Computer Science student and DOST scholar with proficiency in C#, Python, and HTML. Seeking a full-time position in a dynamic and innovative environment where I can apply my programming skills, contribute to impactful projects, and continue to develop my technical abilities as part of a collaborative team.

### TECHNICAL SKILLS

| | |
|---|---|
| **Programming Languages:** | Proficiency in C#, Python, HTML, C++ (Arduino) |
| **Image and Video Editing:** | Adobe Photoshop, GIMP, Filmora |
| **Digital Painting and Illustration:** | Krita, GraphicsGale |

### EDUCATIONAL BACKGROUND

**Tertiary: Technological University of the Philippines - Manila**

Ayala Boulevard, Ermita, Manila

**Bachelor of Science in Computer Science**
2020 - 2024

**Secondary: Las Piñas City National Science High School**

Carnival Park St., BF Resort Village, Brgy. Talon 2, Las Piñas, Philippines

**STEM Strand**
2014 - 2020

# DAN EIAIEL F. EBOÑA

**Email:** dan.eiaiel14@gmail.com | **Mobile No.:** 0999-930-2878

**Address:** 73A Saint John, St. Marlane, Brgy. Minuyan, Norzagaray Bulacan

## CAREER OBJECTIVE

Aspiring to leverage my knowledge to create and develop innovative software and games that provide enjoyable and memorable experience for users. Committed to continuous learning and collaboration to push the boundaries of interactive entertainment.

## TECHNICAL SKILLS

| | |
|---|---|
| **Programming Languages:** | **Proficiency in C#, Python, C++** |
| **Video Editing:** | **Filmora** |

## EDUCATIONAL BACKGROUND

**Tertiary: Technological University of the Philippines - Manila**

Ayala Boulevard, Ermita, Manila

**Bachelor of Science in Computer Science**
2020 - 2024

**Secondary: Centro Escolar Integrated School - Manila**

9 Mendiola Street, San Miguel, Manila City

**STEM Strand**
2018 - 2020

# CYRENE ROSE C. MANIBALE

**Email:** cyrenecmanibale @gmail.com | **Mobile No.:** 0906-278-3476

**Address:** P58-15 15 ST – 16 ST Barangay 183, Villamor, Pasay City

## CAREER OBJECTIVE

Persistent and dedicated to gaining knowledge and practical expertise. Detail-oriented and optimistic with a strong drive to find solutions to problems. With a willingness to learn, acquire new skills, and want to contribute and collaborate with the team.

## TECHNICAL SKILLS

| | |
|---|---|
| **Languages:** | C, C++, HTML |
| **Tools:** | Visual Studio Code, Figma, MS Office |

## EDUCATIONAL BACKGROUND

**Tertiary: Technological University of the Philippines - Manila**

Ayala Boulevard, Ermita, Manila

**Bachelor of Science in Computer Science**
2020 - 2024

**Secondary: Centro Escolar Integrated School - Manila**

9 Mendiola Street, San Miguel, Manila City

**STEM Strand**
2018 - 2020

# MAURICE H. LAGAS

**Email:** lagasmaurice@gmail.com | **Mobile No.:** 0995-759-3313

**Address:** 0765 Sitio Gulod Camalig Meycauayan, Bulacan, Philippines

## CAREER OBJECTIVE

Currently, as a student, my objective first and foremost is to pass and graduate in order to appease and reduce burden from my aging mother. After graduating, my next objective would be to find a local job and support her.

## TECHNICAL SKILLS

| | |
|---|---|
| **Programming Languages:** | Proficiency in C, C++, Java C#, Python |
| **Speaking Language:** | N4 Level Japanese Speaker |

## EDUCATIONAL BACKGROUND

**Tertiary: Technological University of the Philippines - Manila**

Ayala Boulevard, Ermita, Manila

**Bachelor of Science in Computer Science**
2020 - 2024

**Secondary: Araullo High School**

United Nations Ave, Ermita, Manila, 1000 Metro Manila

2011 - 2015