MYTHICA INCEPTION: A Web-based Monster-taming Action-RPG with AI for Dynamic Difficulty Adjustment and Decision-based Storyline

A Thesis Project Presented to the

Faculty of the School of Computer Studies

University of San Jose – Recoletos

Cebu City, Philippines

In Partial Fulfillment of the

Requirements for the Degree

Bachelor of Science in Information Technology

specializing in Multimedia Applications Development

by

Dave Mossess Poro

May 2022

Contents

LIST OF FIGURES, TABLES, AND EQUATIONS

Figures

Tables

Equations

ENDORSEMENT

The project study entitled ‘Mythica Inception’ is prepared by the following student(s): DAVE MOSSESS PORO**,** in partial fulfillment of the requirements for the degree of BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY, has been endorsed and is recommended for the acceptance and approval for ORAL EXAMINATION.

JOHN LEEROY A. GADIANE
Member

JOSEPHINE E. PETRALBA
Member

VICENTE III F. PATALITA
Adviser

JOVELYN C. CUIZON, DMHRM
Chairperson, CS/IT Department

GREGG VICTOR GABISON, DMHRM, DIT
Dean, School of Computer Studies

APPROVAL

Approved by the Tribune for Oral Examination with the grade of PASSED.

JOHN LEEROY A. GADIANE
Member

JOSEPHINE E. PETRALBA
Member

VICENTE III F. PATALITA
Adviser

JOVELYN C. CUIZON, DMHRM
Chairperson, CS/IT Department

GREGG VICTOR GABISON, DMHRM, DIT
Dean, School of Computer Studies

ACKNOWLEDGEMENT

Consummatum est.

A sincere gratitude to all who have helped and supported Mythica: Inception.

It would have never been finished if not for all of you.

ABSTRACT

This project aims to resolve the terminal inadequacies of video games in the genre of Role-playing Games (RPG). The following inadequacies include stale combat mechanics, identical enemy Artificial intelligence (AI) or bots, linear storylines, and ease of difficulty. The project uses real-time combat instead of a turn-based combat mechanic, distinct bot behavior instead of a randomized action for enemies, a decision-based quest system instead of a linear, innate quest progression, and Dynamic Difficulty Adjustment instead of a fixed overworld difficulty to solve the said inadequacies.

KEYWORDS

Real-time Combat, Pluggable Artificial Intelligence, Distinct AI behavior, Scriptable Objects System, Decision-based Quest System, Dynamic Difficulty Adjustment.

CHAPTER I

Introduction

Rationale of the Study

The overall design of a video game, particularly in their Artificial Intelligences (AI), is centered intensely around their behaviors. They are vital perspectives concerning why video games become memorable, but people tend to overlook AI. Unique behaviors that are often hard-coded in every AI are called bespoke behaviors. An example of this is the Super Mario series, where every enemy character has unique behaviors, which makes these characters paramount and recognizable.

Similarly, a video game loaded with content, like Pokémon (1997), which is about taming distinct species of monsters and making them your companion in your journey, is likewise a memorable video game. These types of games are attribute-based. A distinction is that every creature, in this case, a Pokémon, includes various attributes, and by changing each of these attributes for every creature, they will, in general, turn into a unique playable character without changing the core behavior's logic for each. Exploring these topics serves as the researchers' inspiration in doing this study.

Speaking of Artificial intelligence, the researchers will be discussing AI in a video game context or narrow AI, which is vastly different from academic AI. The AI discussed, as Google Arts and Culture (1892) put it, expands the game-player experience instead of decision-making or machine learning. Progressive difficulty levels, unique movement patterns, and in-game events dependent on the player's input made AI prevalent in video games.

Furthermore, the researchers are interested in creating distinct AI in creatures using the bespoke behavior pattern and the attribute-based behavior pattern; in role-playing games (RPGs), specifically the monster-taming genre. The researchers' reason is that the genre handles different attribute-based data much like the series mentioned before, Pokémon, and has different playable and non-playable characters (NPCs) that could put the bespoke behavior pattern to practical use.

While further studying the said genre, the researchers noticed the terminal inadequacies of the video games this specific niche is playing. These include stale combat mechanics, identical opponent AIs, linear storylines, and ease of difficulty. However, these inadequacies existed because of the hardware during the creation of the series. Because of the emerging existence of powerful hardware today, genre changes must be made as this generation's users have a different standard when it comes to the games they play now. The researchers aim to fill in the gaps associated with these genres for years.

Statement of the Problem

As previously said, the association of the monster-taming genre was established years before with a specific franchise. Because of Pokémon, the large franchise which has monopolized the entire genre, the series has become a staple in this specific niche of audience and is what people define as a monster-collecting game. Despite the growing popularity of indie game studios creating these types, such as Crema, Temtem (2020) developer, or Bandai, who developed Digimon (1997), a competitor of the said series in the AAA industry, the target audience in these games often defines these as Pokémon-likes or clones. As a result, the researchers mentioned the inadequacies of

the series above. In addition, the following questions below pose a great deal in this paper:

1. Will a real-time combat mechanic make more compelling gameplay than turn-based combat like Pokémon?
2. With the turn-based mechanic becoming repetitive as in-game time passes, will a flexible distinct AI help these games become less repetitive?
3. A problem persisting in the said series is the game story's linearity, blocking paths because it is not where users are supposed to be. Does giving players the sense of freedom to go wherever they want to help carry out a satisfying story?
4. Does dynamic difficulty adjustment (DDA) help solve the player's complaint about making these types of games too easy while not making inexperienced players feel alienated by the frustrating difficulty of the game?

Significance of the Study

As mentioned earlier, the discoveries made in this study should improve the entire monster-taming genre. The study's goal is to allow the players of various skill levels to enjoy the game without being frustrated by its difficulty level. The game should eventually adjust to the various levels of difficulty based on the player's skill level, whether it is frustratingly tricky or easy. Similarly, the researchers would like to highlight the game's unique story-driven project, which creates different stories for players to tell to other people interested in the genre. Finally, the researchers hope to create an exciting RPG with significant elements of attribute-based game design combined with the bespoke design principle akin to action-driven games, which will eventually leave the turn-based mechanics attributed to the RPG genre.

Project Scope and Limitation

The project's main objective would be to provide all the basic functionality of the RPG genre. The said project includes a skill system, leveling system, items - inventory system, randomness, and the project's strong emphasis on story. All other twisted RPG themes are also welcome as they still fit the project at its best. However, the basic functionalities should be prevalent since niche players will be familiar with them. What distinguishes this project are the key features, which serve as the project's main scope. The project's scope includes the following:

- AI for Dynamic Difficulty Adjustment (AI for DDA)
- Distinct AI Behavior
- Creature-collection/Monster-taming
- Non-linear or decision-based storyline
- Real-time combat mechanics

CHAPTER II

Game Design Document

Game Description

Figure 1: Mythica Logo Design

Mythica: Inception is a monster-taming action RPG. The players can collect and tame these amazing magical creatures called mythicas and strategizing their combat skills by giving the creatures items. The user's mythicas can gain experience by battling other mythicas in real-time—barter items to upgrade them and make them stronger. The players will have to explore Pearl Isles and embark on the grand adventure of a lifetime. They can learn more about the island's story and lore.

Design Goals

The project's creators note the following aims enumerated:

1. Develop an entertaining game by applying Dynamic Difficulty Adjustment to the game's systems.
2. Develop a vast experience for gamers through *exploration*; players explore the Pearl Isles to discover hidden treasures and lore across the map. Gamers can also acquire immense personal breakthrough by *immersion*; the players' decisions are crucial in determining the fate of Pearl Isle and how the story ends. The game's players can also germinate a variety of encounters through *monster battles*; players will use their arsenal of mythicas to battle the

enemies’ mythicas and other tamers that hinder the player’s goals from finishing the game.

Influences and Sources

The entire project consists of influences from a variety of games and genres. Because of the researchers' experiences of playing different genres of games, they have found several video games to be interesting as an influence for the project. The following games are Pokémon Mystery Dungeon Series, Legend of Zelda: Breath of the Wild, Genshin Impact, Undertale, Hades, Diablo 3, and MOBAs.

The project uses the Pokémon franchise as its inspiration with the creature designs. The iconic look of Pokémon comes from their principles in designing creatures to be friendly or to look like they can be players' friends.

Because of its open-world nature, the game uses the Legend of Zelda games, precisely Breath of the Wild. The open-world nature of the game gives the players a sense of free will to do whatever they want.

The researchers took inspiration directly from the character switching mechanic reminiscent of Genshin Impact. Abilities and other game mechanics, such as their world-traversal movements, could also be a potential inspiration to the team, especially for the mythicas.

Undertale's fantastic story and humor could inspire this project as the game's incredible decision-based story give players different stories to tell in the game.

The project primarily focuses on its combat hack and slash with its RPG elements. However, the inspiration for the combat mechanics of the project is from

Hades. This exhilarating game could leave players on the edge of their seats while playing this fast-paced combat game.

The game has influenced the project regarding the skill progression system and its game design philosophy as a dungeon crawler, hack-and-slash, action role-playing game.

The researchers are deeply fascinated by the MOBA genre's item-crafting mechanics. These give players a sense to strategize how to equip their characters in the different MOBA games. So, the researchers would like to give this twist in its RPG mechanics by making players think about what items should be in their different tamed mythicas.

Target Market

The researcher’s ideal user base has the following characteristics:

1. Male or female;
2. 15 to 25 years old;
3. Likes adventures;
4. Likes games;
5. Likes RPGs;
6. Likes collecting;
7. Likes open-world games;

Functional Specifications

Game Mechanics

Core Game Play

The Playable Character (PC) moves in a three-dimensional (3D) plane on a top-down perspective exploring different tribes and cultures that extend from up, down, right, and left of the screen. Wild mythicas (monsters) appear along the way as the characters try to fulfill quests and main story goals. The PC may defeat or tame these wild mythicas with the help of the PC's tamed mythicas. Defeating them may let the player's mythica gain experience points to level up and make it stronger. Along the PC's journey, encounters with local mythica tamers which challenge the PC to beat his/her team may occur, and the PC has an option not to battle them.

The PC's whole journey in the isles stemmed from a renowned tribe leader giving her a task to help her with her plan to achieve a complete human-mythica harmony. As the PC progresses across the tribes, specific actions and events will trigger achievements with gold or items as rewards. Gold is the in-game currency that allows players to buy items that make their party stronger. Some items cannot be bought but only bartered with other items.

Dialogue scenes related to gameplay are on display during gameplay through the HUD. Cut scenes are also present, advancing an overarching storyline. The PC's main task is to convince at least seven tribe leaders in the isles to help the renowned tribe leader with her plan. To convince the tribe leaders, they may give the players a quest or defeat them in a mythica battle. Tribe leaders who challenge the PC in a mythica battle may have to battle all the tribe leader's officials first before battling the tribe leader.

Game Flow

There are actions that the PC can perform. The PC can move left, right, up, and down. They can also dash left, right, up, and down. Releasing a tame beam to fill the tame value of the wild mythica is also one of the actions the PC can execute. They can also switch from different mythicas in your party. Another action that the PC can do is performing the basic attack of the current mythica. The PC can also talk with NPCs and grab dropped items.

Characters / Units

There are also specific units with which the PC can interact in the overworld. Eleo/Luna, the cannon name of PC, washed up on the shores of Tribe Opón, determined to start a revolution towards the highly urbanized city of Alpas to bring harmony between humans and mythicas. Wild mythicas can also be one of the units. They are unique and powerful creatures that inhabit the world.

Since dawn, these creatures have been captured and tamed for humanity's ventures. NPCs can also interact with the PC since they give information about the world and the isles the player is in and will give directions or hints about the world. They may also give side quests. The PC can also encounter tribe leaders. They are bosses that challenge the player or give them tasks/quests to convince them. The player's goal is to harmonize between mythicas and humans so they will completely live alongside every human being in the world.

Game Play Elements

In the project, some elements communicate with the player's game state. These are namely, the Health bar and Dialogue scenes.

The Health Bar symbolizes the player's or their mythica party member's health. The health bar's value depends on the mythica or the tamer's health. When all their mythica party member's health depletes, they will faint, and an automatic switch to another mythica in the party will occur. If all the party members faint, it will automatically switch to the PC. If the PC faints, they will return to the last healing center they stopped by.

The Dialogue scenes will be boards representing the communication exchange with text shown to the player on the HUD during interactivity. The player will get essential data through the discourse.

Some components decide the PC's statistics: the Experience Points and the Level.

A mythica takes in experience points whenever it ends a foe and executes quest tasks. Once a mythica has earned enough points, it adds its overall level. The level displays a mythica's advancement, as it gains a level once it earns sufficient experience points.

Game Physics and Statistics

Physics

The creators based the game's physics on the game engine's physics engine. To describe the game's general physics, one of the factors to enumerate would be that the PC moves in a 3D top-down perspective, either from left to right or vice versa or from up to down or vice versa. The PC cannot cross paths with enemy characters and have no pushbacks when hit by an attack. However, enemies such as wild mythicas, mythica tamers, and tribe leaders will have no push back when hit by an attack. The PC cannot

proceed further in the direction where an obstacle is in front of the way. Lastly, there is the existence of gravity.

Stats

Stats are any specific numerical values about each mythica. The project uses the mythica's stats in combat and world traversal. Stats are short for statistics; in some cases, they have also been named ability, rating, effect, or parameter.

One of the mythica's stats is its **Hit Points** (HP). It determines how much damage a mythica can receive before it faints.

Another stat is the **mythica's life limit**. It determines how much a mythica can faint before it completely dies. The mythica's life limit decreases every time it knocks out. Once the mythica hits less than 0 lives, they will permanently die. The Life Limit fits well in the game as the theme focuses on the narrative that mythicas are stoppable creatures, and they, too, suffer from the abuses of humankind. It also gives a sense of care towards the player's tamed mythica. It is the player's lifeline to traverse the world and complete the story thoroughly.

The **Base Experience Yield** is also one of the mythica's stats. It is a factor in determining how many experiences points a knocked out mythica will give to the mythica that defeated it.

Its **Stability Value** is also one of a mythica's stats. The value makes players' mythicas unique. Each mythica the player encounters in the wild will have a random stability value from 1 to 50. The value makes the same species of mythica different from each other in terms of combat. The stability value affects the game's combat mechanics as the higher the stability value of the mythica the player captured, the stronger player's

mythica is.

Tame Resistance (TR) is another stat of a mythica. It determines how many Tame Beams a mythica can deal with before being captured or tamed. A Taming Staff is used to tame a mythica which shoots Tame Beams in which it should accumulate the Tame Value of the mythica to be tamed and be part of the player's party. Values range from 1 to 255.

One of the typical stats in monster-taming games is the **Physical Attack** (Attack). It partly determines how much damage a mythica deals with when using a physical skill. Values range from 1 to 255.

Physical Defense (Defense) is also a specific stat in monster-taming games. It also partly determines how much damage a mythica receives when hit with a physical skill type. Values range from 1 to 255.

Special Attack is another stat common to monster-taming. It partly determines how much damage a mythica deals when using a skill. Values range from 1 to 255.

Special Defense partly determines how much damage a mythica receives when hit with a skill type as a specific stat in monster-taming. Values range from 1 to 255.

One stat common to action games is **Critical Rate** (Critical Chance). Expressed as a percentage determines the probability that an attack will damage significantly and deal extensive bonus damage. By default, 5% will be the lowest possible critical rate.

A stat vital to gameplay, **Movement Speed** is the speed a unit can move over a second.

Lastly, a common stat of mythica is the **Attack Type**. It determines whether the mythica usually does melee attacks or ranged attacks.

Artificial Intelligence

As said earlier, artificial intelligence in the context of video games are described in this passage.

To describe the characteristics of the AI in this context, Wild Mythicas are one of the AIs that appear in-game. It may appear on-screen from the right, left, up, or downside. Specific mythica species will be in specific places in the overworld. Specific mythica species may pursue the PC on sight or wait until within a specific range. Others may run away from the PC to avoid getting tamed. Those mythica species that pursue the PC may attack as soon as they are in their field of view. Their attacks will vary, but they will be able to attack repeatedly. However, if using skills, they will be unable to use skills on cooldown. Like the PC, the wild mythica also has stamina, limiting the mythica from using skills too much. Once the wild mythica's health bar is low, it starts to flee from the player until it is out of its field of view.

USER INTERFACE

Flow Chart

It is important to note that having a flow chart for the game's user interface would guide the researchers to have a smooth user experience. Below is the flow chart of the game's user interface.

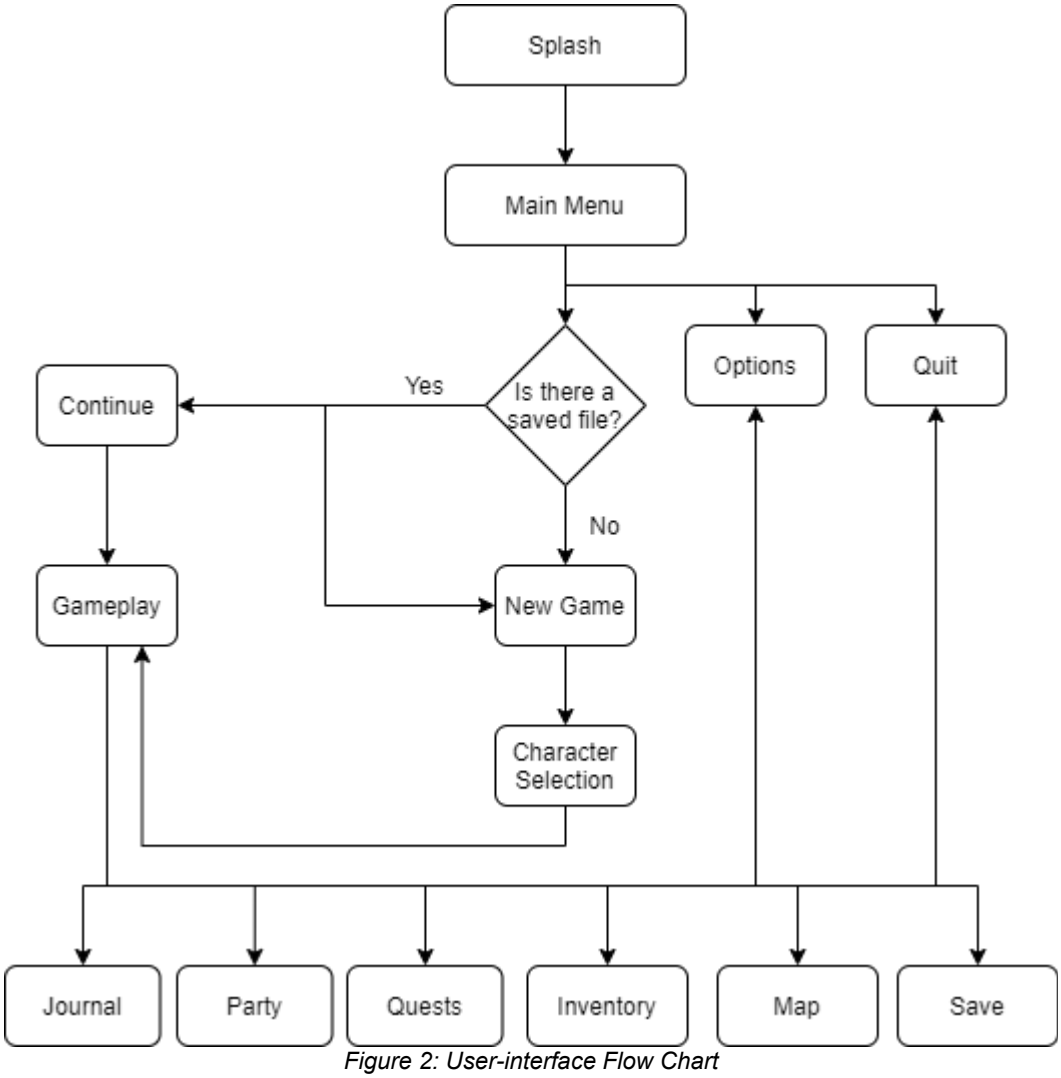


Figure 2: User-interface Flow Chart

Functional Requirements

By describing the different elements found in the flow chart, the researchers enumerate the different elements and their definitions.

One element found in the flow chart is the Splash Screen. It is the initial screen and introduces Blacklight's, our team, logo.

Another is the Main Menu. It presents all the possible choices for the user and a captivating graphic image that introduces the look and feel of the game and the series.

One element is the New Game / Character Selection. It shows the character selection screen where the user can choose whether the players are male or female. The Game/Character Selection Panel is also where the players will input their character's name.

Another element is the Continue. It displays all the user's game states and gives the option to continue from where they last left off.

The Options panel is one of the elements. It gives the user the ability to control some of the game's attributes, such as sound effects and music volume.

Another element is Quit. It gives the user the option to exit the game

Gameplay is also one of the elements in the flow chart. It is where the game happens. It also has a fixed Heads Up Display (HUD) containing the current switched character's portrait, health, experience points, skills, equipped items, the player's mythica party, each party member's health, and the minimap.

Another element, part of the gameplay, is the Journal. It is a page where the user can see what species of mythicas they have seen and tamed.

Party is also one of the elements in the flow chart. It is a page where users can

see their current mythica party, skills, and information.

The Quests panel is also one element in the flowchart. It is a page where users can see what quests they have accepted and keep track of the main quests.

To conclude, one element in the flow chart is the Inventory. It is a page where the user can see what items they possess and which items are equipped by a party member.

Mockups

Before designing the game's user interface, mockups should be created to have a short glimpse of what is to come for the game's UI.

1. Splash Screen

Figure 3: Splash Screen UI Mockup

2. Main Menu

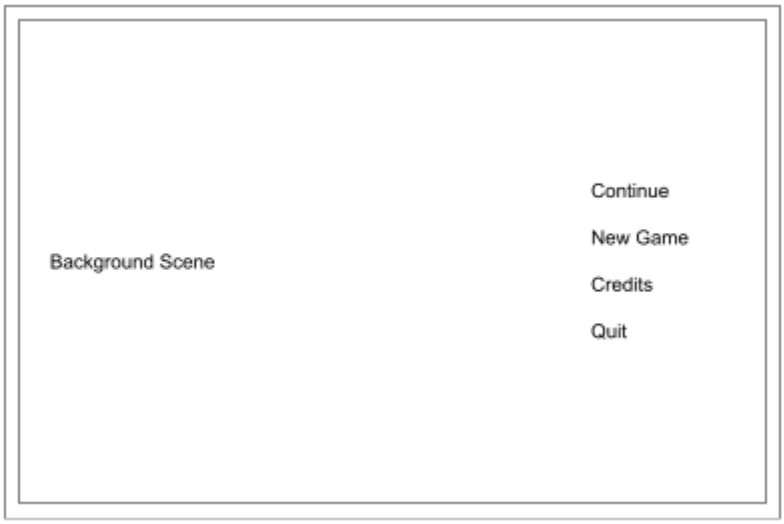


Figure 4: Main Menu UI Mockup

3. Credits

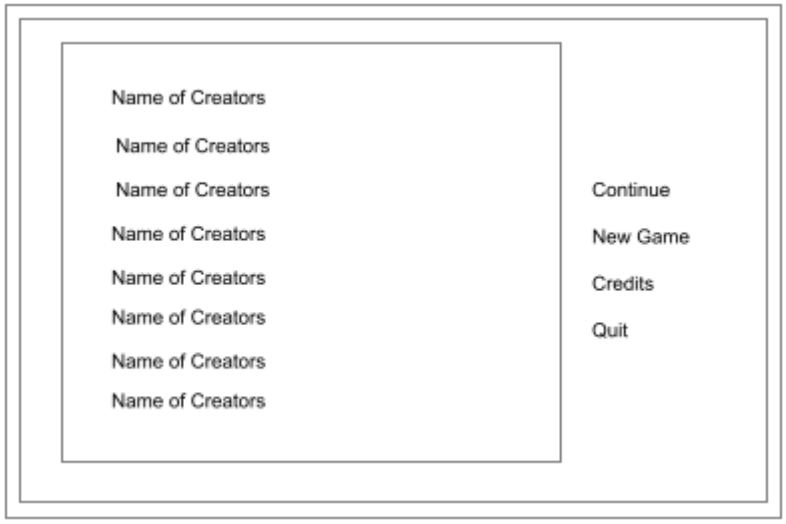


Figure 5: Credits UI Mockup

4. Character Selection

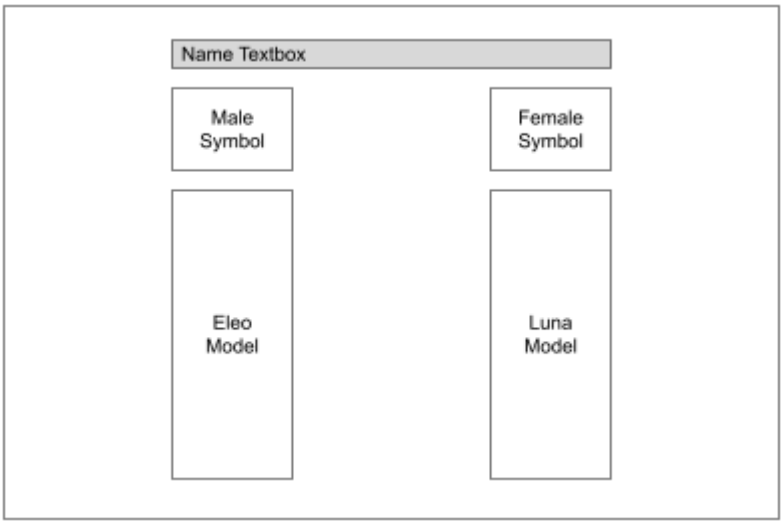


Figure 6: Character Selection UI Mockup

5. Journal

Figure 7: Journal UI Mockup

6. Party

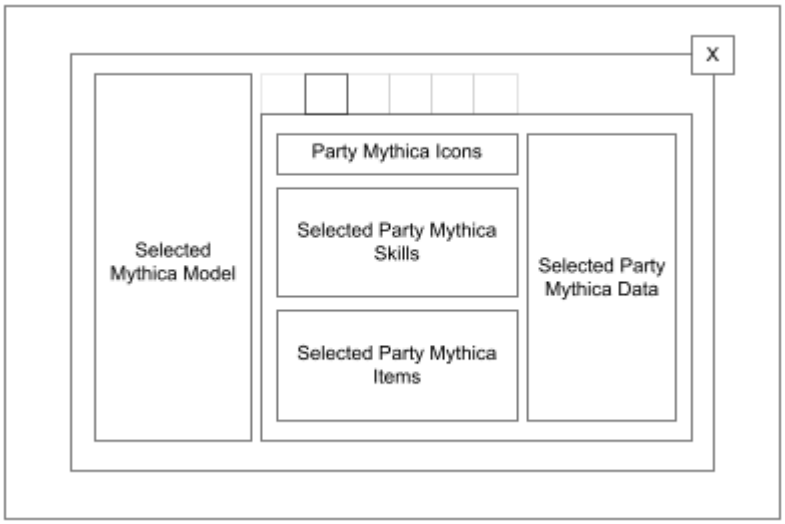


Figure 8: Party UI Mockup

7. Quests

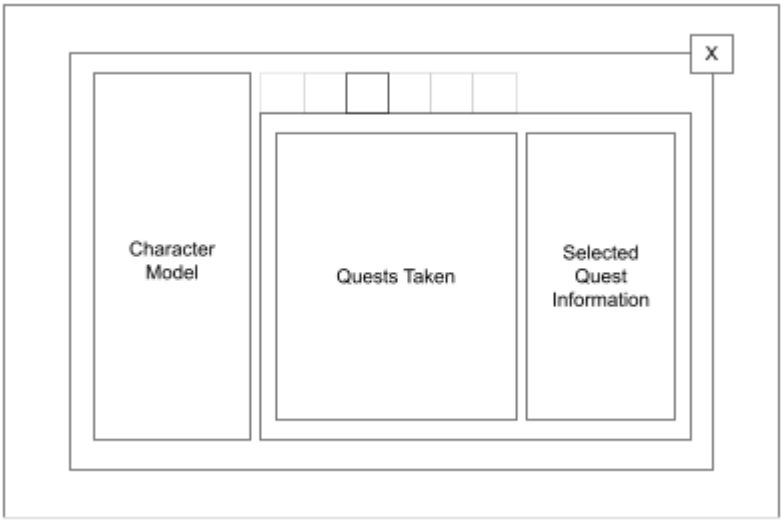


Figure 9: Quests UI Mockup

8. Inventory

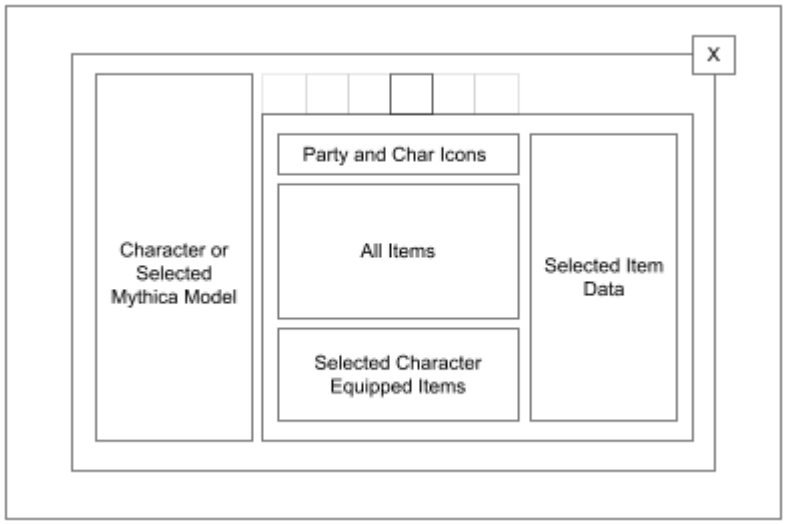


Figure 10: Inventory UI Mockup

9. System

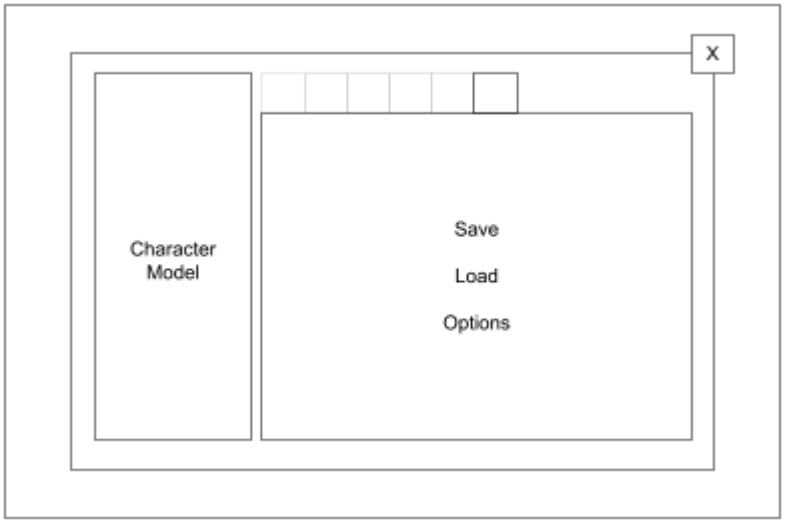


Figure 11: System UI Mockup

10. Gameplay

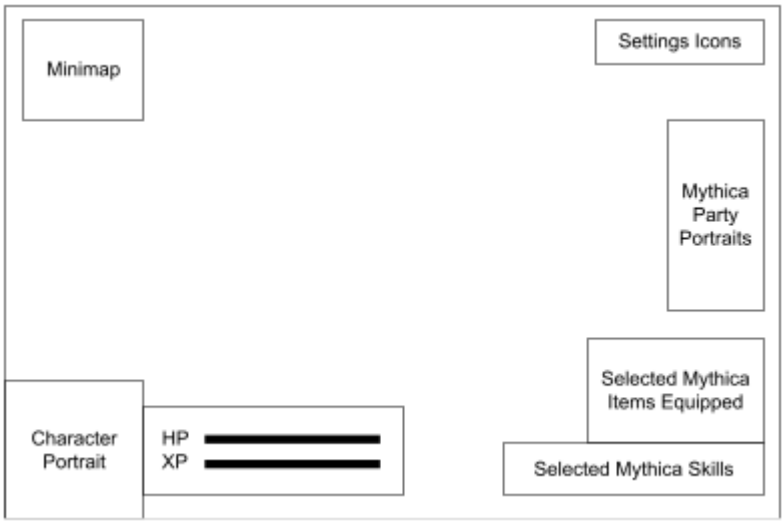


Figure 12: Gameplay UI Mockup

GUI Objects

The game divides the GUI into a few sections. First, we have the Heads Up Display. The HUD is the information on the screen when the user is playing the game. The HUD contains the following: the Minimap, the Character Portrait, the Character's Current Data, the Mythica Party Portraits, the Selected Mythica Items Equipped, and the Selected Mythica Skills.

The Minimap displays the player's current location, which can help find the player's way around the isles.

Character Portrait displays the selected character's portrait, the mythica, or the player. The character portrait is an add-on touch so players would get to know the currently selected character a lot more.

Character Current Data displays both the HP and XP of the currently selected character, a mythica or the player.

Mythica Party Portraits shows the player's party mythica and their HP so players would know what mythica to switch during combat.

Selected Mythica Items Equipped shows the currently selected character's items equipped so players would be able to know what items the player is equipping with during combat.

Selected Mythica Skills shows the currently selected character's skills so players would be able to know what skills to use during combat.

Art and Video

Overall Goals

We have sought to achieve a friendly, bright, and stylistic art style from the beginning. However, this style will develop over time as the story becomes more severe and progresses. This cartoony yet stylistic polish will entice players of all ages.

3D and 2D Art and Animation

We have extensively used original artworks designed by the researchers, from the UI elements and 2D mythica designs to their respective 3D models. With the help of a Toon-Shader made by the researchers using the game engine’s built-in shader graph for the characters, we intended to achieve an art style that reflected the researchers’ inspired visuals but still provided a fresh and unique experience for the player.

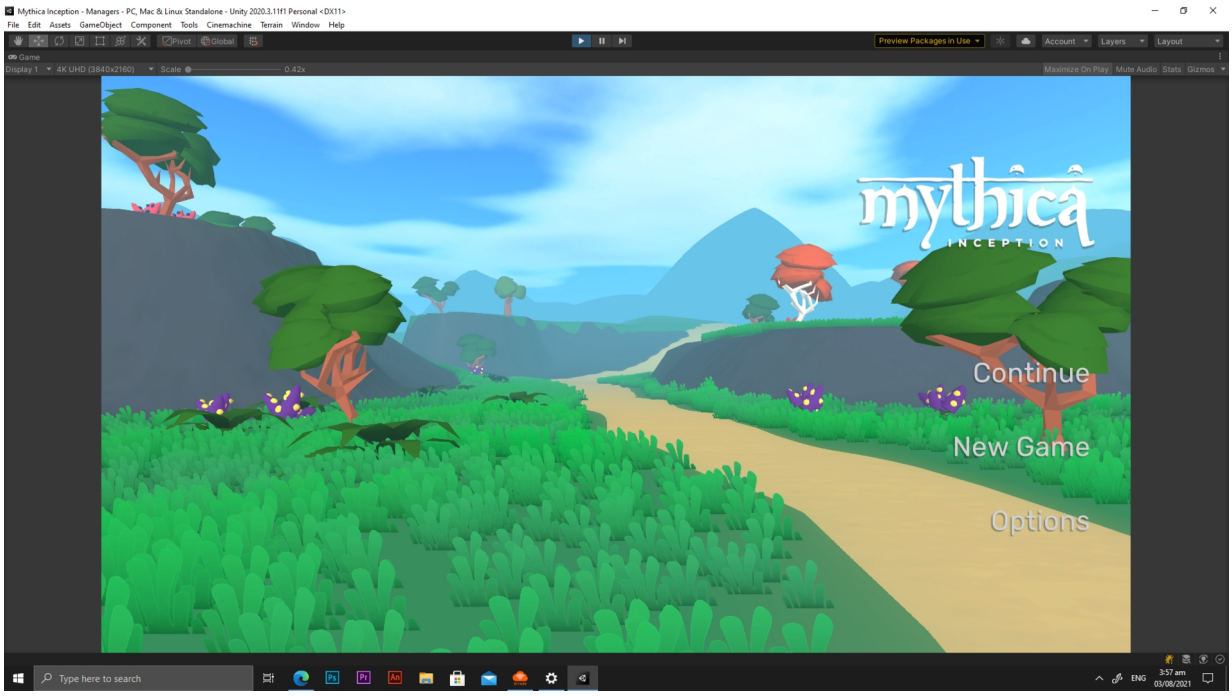


Figure 13: Main Menu Screen Capture

Figure 14: Player Characters Design: Eleo and Luna (Design Concept)

Figure 15: PC: Luna 3D Model

Figure 16: PC: Eleo 3D Model

Figure 17: Mythica Design: Chanaque (Design Concept)

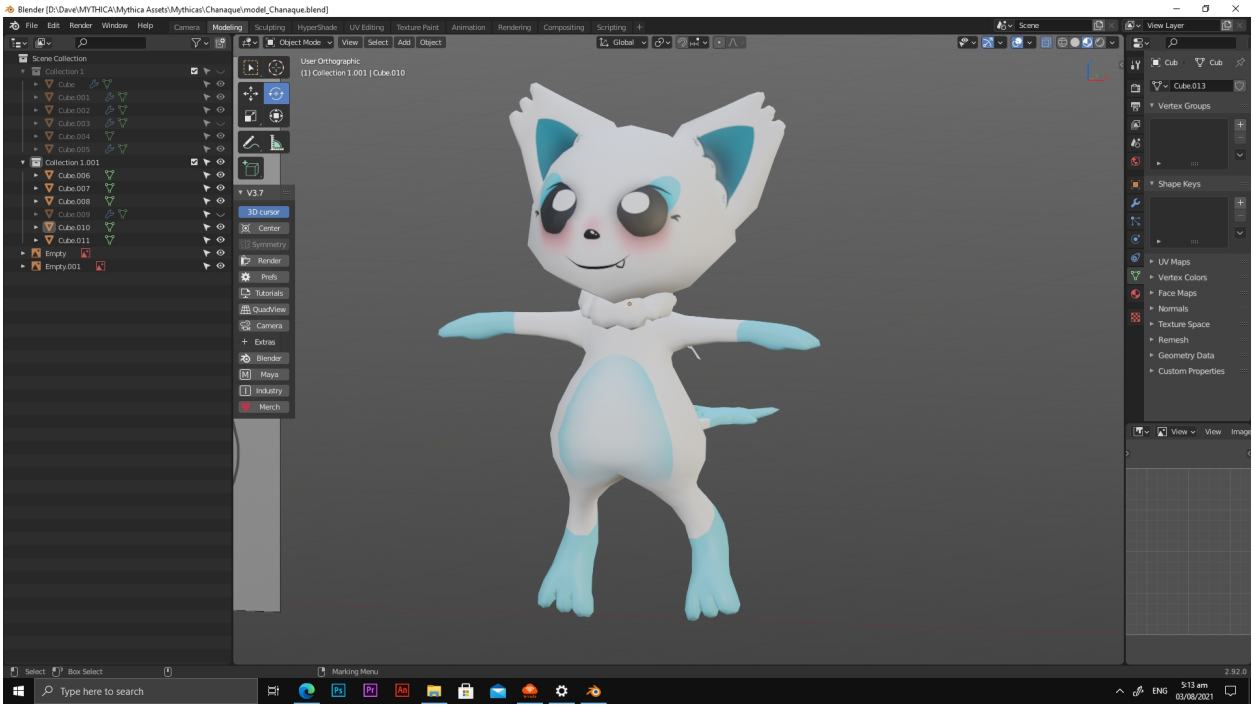


Figure 18: Mythica Design: Chanaque (3D Model)

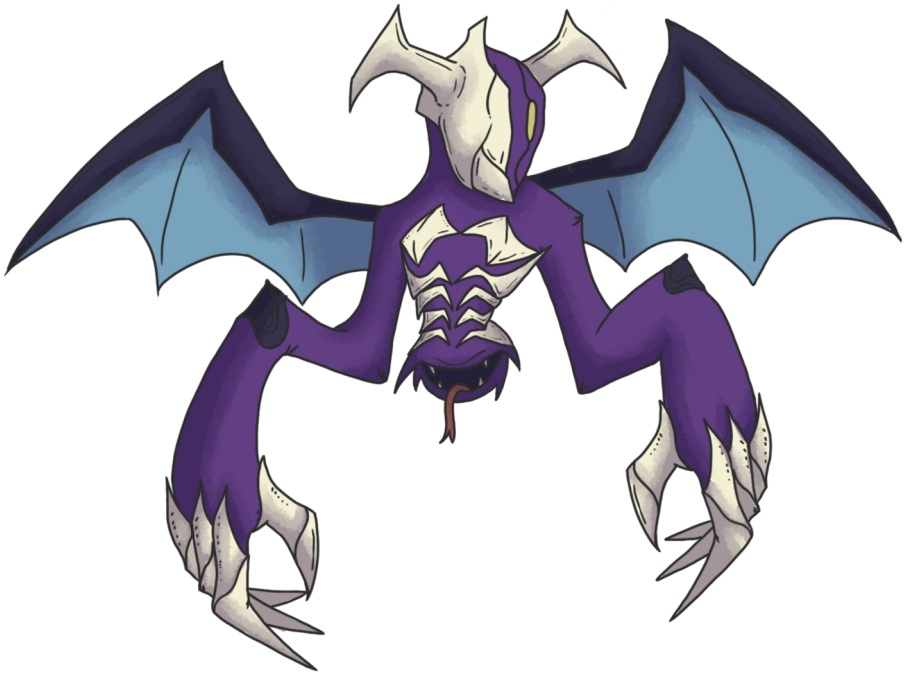


Figure 19: Mythica Design: Separaghoul (Design Concept)

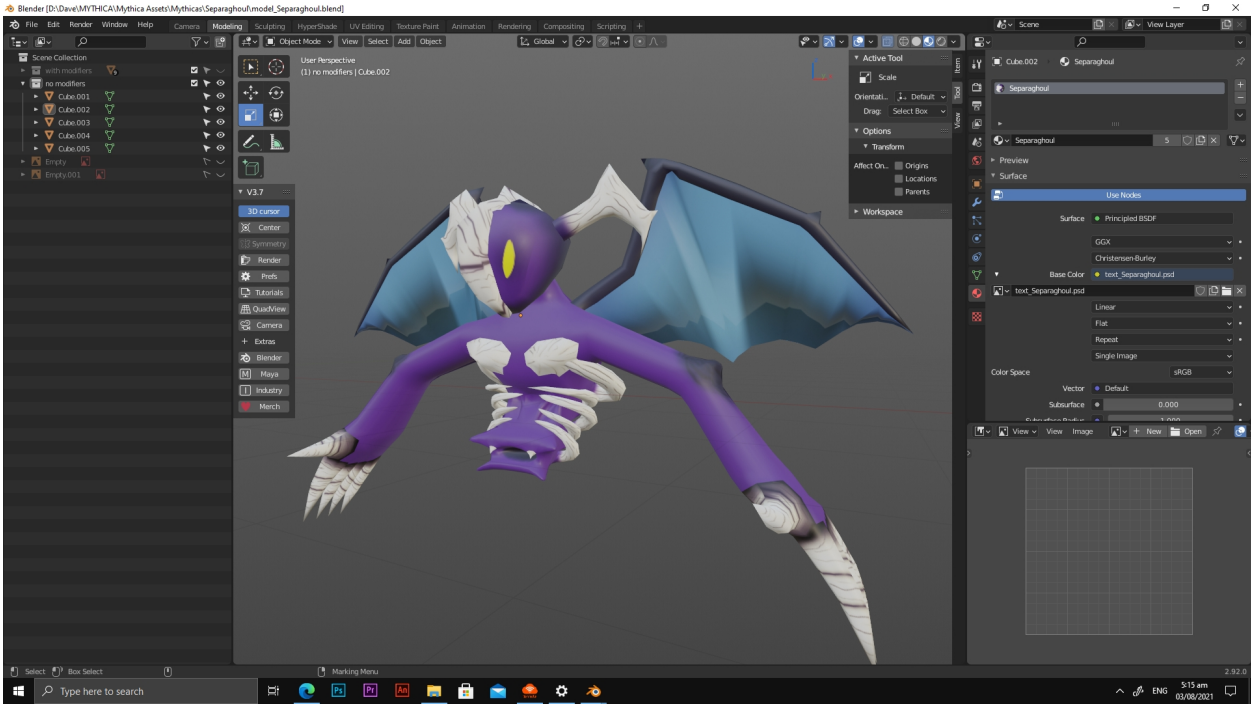


Figure 20: Mythica Design: Separaghoul (3D Model)

Figure 21: World Design: Pearl Isles Map

GUI

The critical design philosophy in terms of the game's UI uses rounded quads and minimalistic designs to achieve a friendly look and feel to the game. Since the game's intended audiences are the Nintendo Co., Ltd. audience, which intends to deliver family-friendly games while not alienating older players, this design philosophy would help invite the target audience to try the game themselves.

Marketing and Packaging Art

The marketing materials for the game consist of having our game icon, which is the symbol found in the project's logo. The symbol is reminiscent of the yin and yang symbol; however, the symbol means that there is no black and white to every story, but there is a gray area which means even if people have good intentions, they could also have damaging schemes behind them. The cover art and packaging are also present,

which displays a clear night sky with the taming staff used to tame the mythicas present in the world. When one opens the game, the Splash Image, the first image with the Blacklight Games logo on it, is the research team's logo.

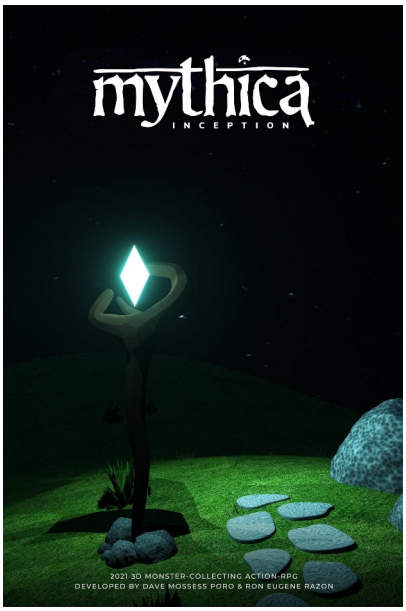


Figure 22: Cover Art and Packaging

Figure 23: Splash Image

Figure 24: Game Icon

Terrain

The used game engine's built-in terrain tools help with the terrain-making process. However, planning its visual design elements is still essential in achieving the researchers' intended uniformed look and feel for the game. Using plain colors for the terrain and importing it with the lowest possible quality compression will help the porting process to a WebGL format, which the researchers intend. However, using a subtle touch of normal maps will help give the terrains some depth.

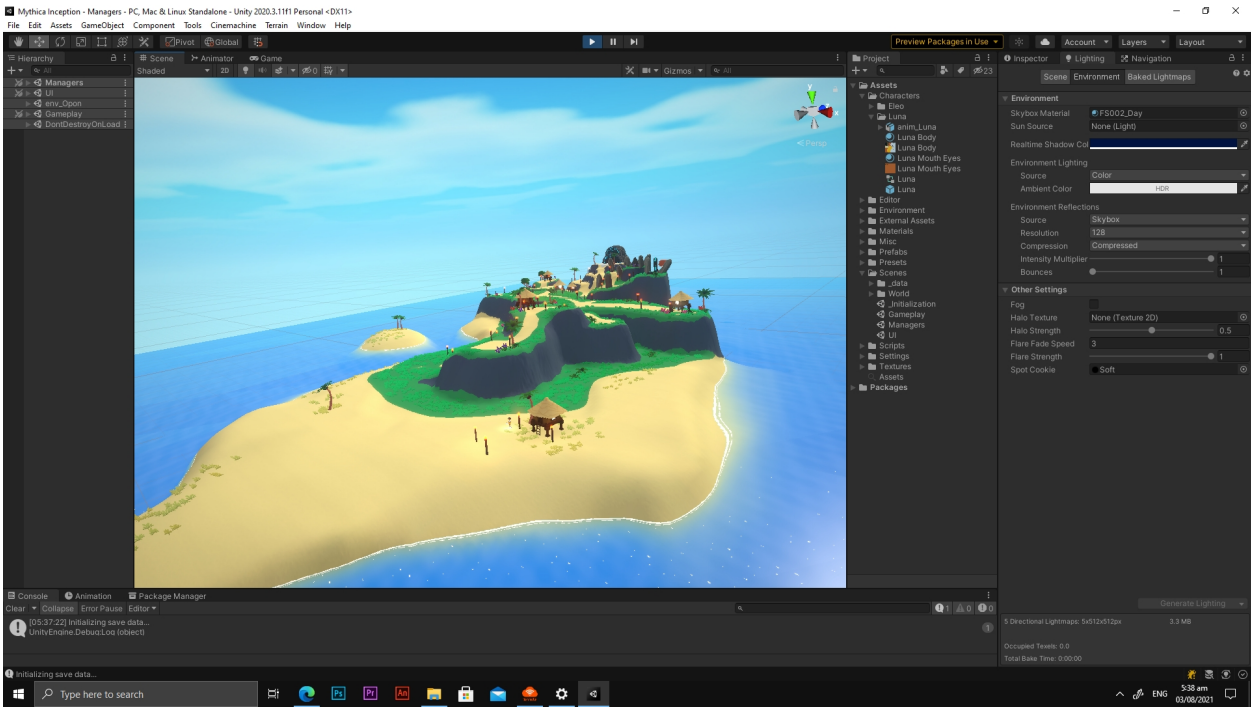


Figure 25: Tribe Opon (from afar)

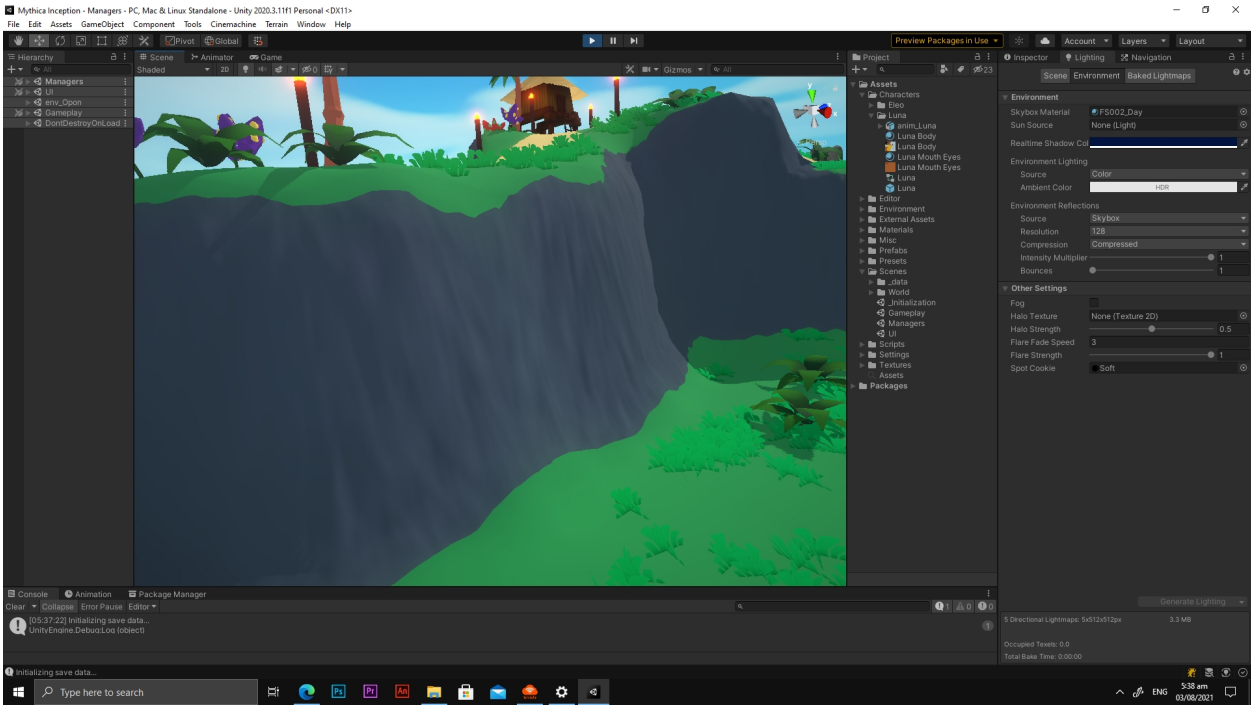


Figure 26: Tribe Opon (normal maps)

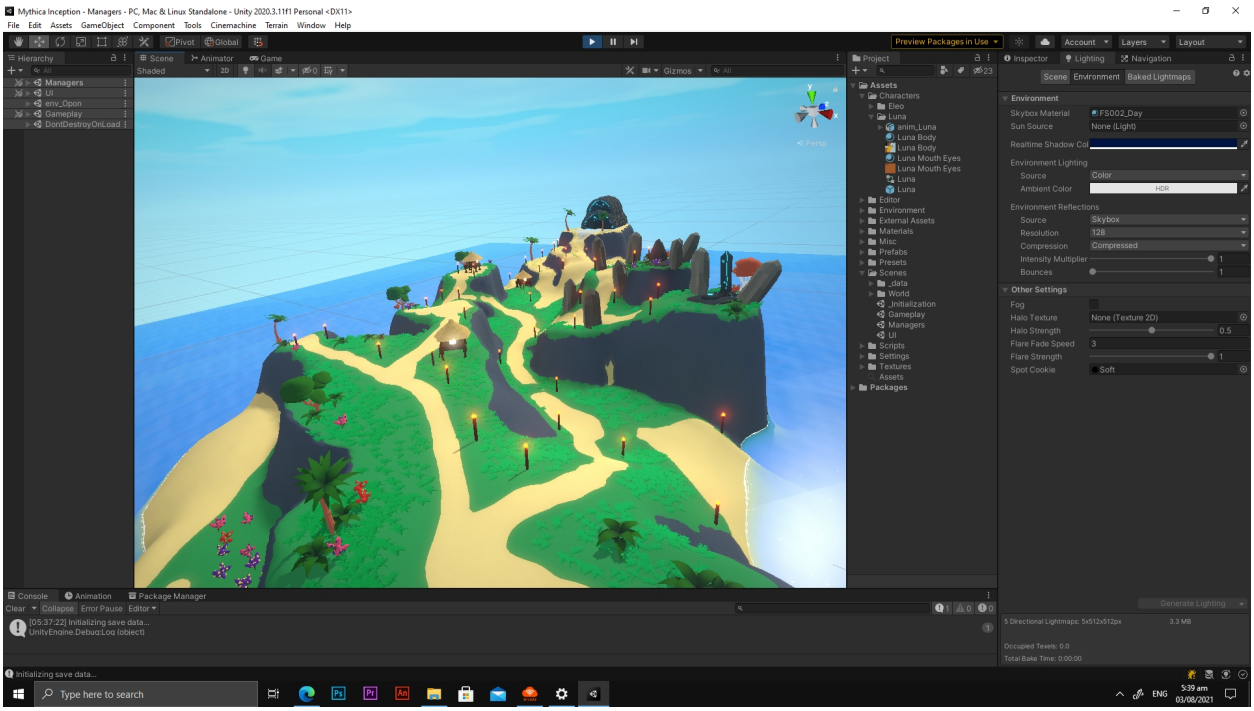


Figure 27: Tribe Opon (up close)

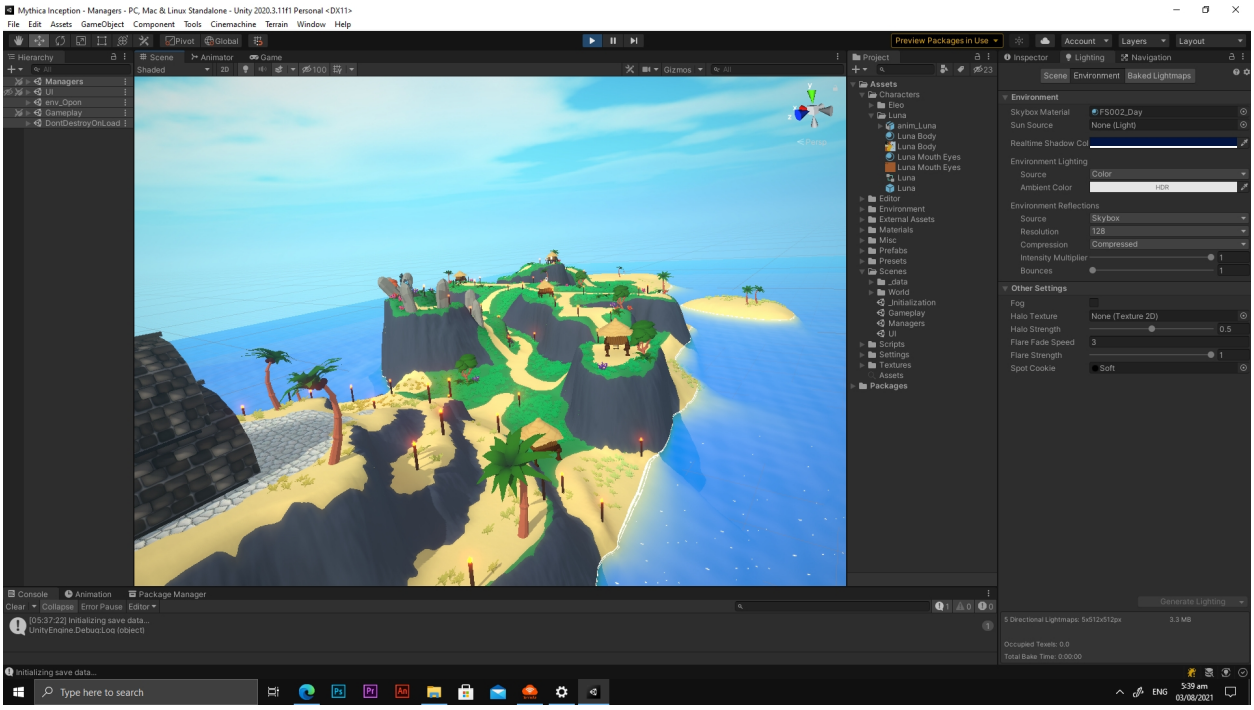


Figure 28: Tribe Opon (view from north)

Sound and Music

Overall Goals

The goal of the audio in the project is to set the world's mood and indicate a call to action for the player during gameplay events. There are three classifications of audio in the game: Music, Sound FX, and Ambiance. Music will be requested by Mood, Situation, or by Title. The Audio Manager can request by the name of the sound, and its pitch can be randomized. It can also be in a specific pitch. The project can request Ambiance by name. Some objects will have an ambiance attachment, which means a GameObject has an Audio Source component with a Spatial Blend of 3D. The player will only hear the object with this ambiance attachment if they are near its location.

Sound FX

The sound effects of the game will be from a site called freesound.org. It is a collaborative database of open-source and royalty-free sound effects assets. Then, the researchers will process the Sound FX from Audacity. Audacity is a free and open-source, cross-platform audio software to give it a slightly different tone than its base sound effect from Freesound.

The game’s dialogue sound is also part of the game’s sound FX. The dialogue’s mechanic is also the same when requesting sound FX in the system. There is a specific sound for each letter. The system will play the specific sound-letter whenever the dialogue system shows that specific letter on the screen.

Music and Sound FX Assets

As stated, background music used in the overworld will depend on the situation or mood. Each song will have its situation and mood in the game. The system will

randomize the processing of choosing music whenever the game requests it. The following music is the in-game soundtrack for the project:

Artist: **Kevin Macleod**

Website: **www.incompetech.com**

Licensed under Creative Commons: By Attribution 4.0 License

http://creativecommons.org/licenses/by/4.0/

Music:

Anamalie	Easy Lemon	Swimmey Texture	Porch Swing Days – slower
Avec Soin	For Originz	Cattails	Take a Chance
Danse Morialta	Fresh Air	Eastminster	Late Night Radio
Earnest	Immersed	Impromptu in Blue	Perspectives
Ether Vox	Silver Blue Light	Space X-plorers	Windswept
Gagool	Summer Day	Healing	

Artist: **Leviathan Music**

Website: **soundcloud.com/leviathan254**

Licensed under Creative Commons: By Attribution 1.0 License

http://creativecommons.org/licenses/by/1.0/

Music:

Merry Quikmas

Precision

Rocking Waves

Royalty Free RPG Battle Theme

Royalty Free RPG Battle Theme #2

Royalty Free RPG Battle Theme #3

Royalty Free RPG Battle Theme #4

Royalty Free RPG Battle Theme #5

Story

Player Character

Player Characters are the characters often seen by the players. The player characters need to have a fleshed-out story for players to feel their importance within the game.

Eleo and Luna are the cannon names of the Player Character. On PC's 18th birthday, Carlyle is ecstatic for her son/daughter to have a whole future already set up, being one of the future courageous soldiers of Alpas. This elite organization of heroic soldiers aims to bring peace and order to all of Pearl Isle. With this, it plans to bring its advocacy of liberalizing the mythicas in the isle to reality. This plan separates the mythicas from humans, so mythicas do not have to suffer from the innate havoc humans bring to their habitats and their world. However, their entire purpose changed on the day of their 18th birthday celebration. Before PC's 18th birthday celebration, the right age to be on duty as an Alpas soldier, they saw an old portrait of Carlyle with her partner mythica, Separaghoul. This portrait destroys PC's dream of becoming an Alpas soldier as it betrays their passion for it.

Outraged by what PC saw, they question Carlyle why she has this portrait and has lied to them ever since they were a child. Carlyle tried to calm PC because they might be heard outside their room, but PC continued their rant. PC continued to explain that they secretly liked the idea of mythicas living alongside humans, and they wanted to have a partner mythica on their own. After a few minutes, a group of Alpas soldiers opens the door to their room and arrests Carlyle. Because PC is skilled at combat, they escape, gliding away from the soldiers. However, one of the soldiers shoots PC and

shocks them, making PC unconscious. Then, a sudden eclipse happened, and the whole city darkened for a moment. A few seconds later, the soldiers were shocked as the PC disappeared that moment. PC is now outside Alpas, washed up on the shores of Tribe Opón.

Since this is the main character, MC will not have a distinct personality.

Enemy Characters

The stories of the enemy characters are essential for the game to balance the story completely.

Rajah Suláymán is one of the enemies in the game. The most feared tribe leader of Pearl Isle, she ordered her partner mythica to kill Lumáyá's partner 30 years ago ruthlessly. She is also the most long-reigning tribe leader in the aisle as nobody wants to challenge her, afraid of their mythica killed. However, that was 30 years ago when she was younger. She is a kind-hearted, gentle-talking, silly character as seen in public. However, nobody knows her true intentions and nature because she always sits on her throne only to go out whenever she walks with her partner mythica.

Her personalities are silly and kind-hearted. She talks gently but often makes sarcastic jokes. However, she is evil, ruthless, and hungry for power when nobody is around.

Another antagonist is Lord Sigmund XXXIX. He is currently Alpas' ruler. He controls everything in Alpas and has been the heir to an extensive line of the city's political dynasty. Alpas' system may be democratic, but their family will always win the elections. During his reign, he has been a terror to the tribes as he abuses the people to get the city's resources. He unleashes a large amount of military power every day

outside of the town to make tribes give what the city needs daily. Now, he wants a systemic change. He wants to be the first ruler of all of Pearl Isles and will one day rid the tribe peoples' control over the mythicas, one thing his ancestors have not done yet. He plans to capture the beast, Adlakawa, which could potentially have the power to separate the mythicas from the humans as it is the Mythica God of Liberation itself. One thing he needs to do is to tame the beast and convince it to separate mythicas from the horrors of human abuse. He knew this because a soldier, one day, brought a scroll, retrieved it from one of the dungeons scattered throughout the isles, and read a part of Adlakawa's power.

His personality is profound, sinister, dictator-like. He is also hungry for power.

Story Theme

The story centers around the existing ethics surrounding the world of mythicas, especially in the Pearl Isles. Its goal is for the player to see whether it is ethically correct to tame these mythicas, use them for every human activity, and make them the player's acquaintances for human accomplishments. Its other goal is for the player to see if it is for the greater good to separate these fascinating creatures from humanity to liberate them for the last time.

Visual Theme

It centers around a pre-colonial place of trade and barter for tribes to start a revolution against the urban landscape of Alpas, a city trying to urbanize the pre-colonial tribes of the isles. Fantastical creatures called mythicas are everywhere in the wild, and the culture of capturing and taming mythicas is the norm.

Story Outline

At this point, the creators of this project refer to the player character as PC, not by their cannon names, since players can freely choose their names and character.

Arc 1 – Harmony

- **Dream**

PC dreams of Buwánáwá, one of the Keepers of Alpas, representing the deity of harmony and earning a starter mythica.

- **Suláymán**

PC washes up in Opon and sees an older woman. The older woman talks to PC and asks for a favor. After the favor, the older woman introduces herself as Rajah Suláymán. She gives PC a brief introduction to the tribes outside Alpas and talks about the isle's current situation, further explaining more information about the isles and Alpás.

PC's disappointment towards her mother fades, and they become eager to open Alpas and its people to the mythicas. Suláymán proposed a bargain to PC. Suláymán and her tribe will help PC fight the soldiers of Alpas to retrieve PC's mother if PC convinces all tribe leaders of Pearl Isle to break the barriers between humans and mythicas in Alpas. Suláymán tells PC to go to the main grounds of the tribe.

- **Tribe Ópón**

PC arrives at the main grounds of tribe Ópón and observes the setting. PC learns that the tribe is focusing on combat and physical enhancement. PC wonders why their tribes accept mythicas as part of their lives. Mythicas serve as their partners or sidekicks in combat. Lákán Dulá, the tribe leader, approaches PC. PC talks about the tribe and then

tells Suláymán's plan. Lákán Dulá challenges PC to a mythica battle which shows a test of strength and bond between the mythica tamer and their tamed mythicas. The tribe leader will help with Sulayman's plan if they complete the challenge. PC defeats the tribe leader and develops a deeper bond with their mythica. Lákán Dulá, in defeat, explains that a human and a mythica can have good chemistry, which makes them in tandem during combat. They wonder if that is why Carlyle, their mother, had a mythica herself. PC finds a wild mythica washed up on the shores of Opon, together with a young woman trying to save it. PC goes to the mythica and tries to help with the situation. After PC rescues the mythica, the young woman trying to save it introduces herself as Ligáyá, and she is an aide and distant relative of Sulayman. She explains that Sulayman sometimes goes to Opon to get away from the stress that her tribe gives her. The PC learns it is a mythica called Alan from Ligaya. The Alan mythica tells them that she and her tamer were separated while fishing, and she needs to be in Áltó with her tamer. Ligaya volunteers to help the mythica since PC already helped Alan.

- **Tribe Áltó**

PC quickly notices that tribe Áltó is different from the first tribe. Unlike tribe Opon, tribe Áltó focuses on agriculture and fisheries. The mythicas work co-dependently with humans in livelihood. Mythicas utilize their unique abilities in farming, fishing, and breeding. PC is amazed by how mythicas and humans work side by side. PC meets Alan's tamer and brings them to the tribe leader, Datu Sándáy. Sándáy talks about how the sales of their produce declined. It was due to the recent ban of their goods in Alpas. Alpas is their major reseller. PC tells Sanday about Sulayman's plan. Sanday is doubtful but then realizes that their tribe can enter the Alpas market again if they take down the

current ruler. Sanday asks PC to retrieve java plum seeds in the dungeon on the small island near the tribe. After completing the task, Datu Sanday accepts the PC's proposition. Sanday gives PC products that would boost his/her/their mythicas. Sanday then asks another favor from PC to deliver some goods to the tribe leader of tribe Íráyá.

- **Tribe Íráyá**

PC arrives at the festive and busy tribe Íráyá. Somehow, the mythicas here serve as personal bodyguards. Mythicas are protective and caring towards their humans. The tribe is currently celebrating the occurrence of a total solar eclipse, which only happens every 500 years. People believe this event is the return of Mayari from the heavens. The world becomes dark for a moment so that no one can see what form Mayari takes during her comeback. Mayari steps down on earth with a mission to bless the people with abundance, peace, and harmony. A lunar eclipse ten days before the event will signal the people to know about her return. Ancestors wrote this prophecy in dozens of scriptures scattered around the Isles, but other tribes think that this is just a myth. Only tribe Íráyá believes and honors the greatness of Mayari. Along with festive food and decorations, the tribe prepares a rare kind of meat as an offering to the deity.

PC asks around about the whereabouts of the busy tribe leader. PC meets the jolly and excited Ligaya. She is in Iraya to enjoy the festivities in the tribe. PC meets Rajah Tupás, the tribe leader, and gives the goods from Datu Sanday. PC learns that the good's purpose is consumables to celebrate their festival. Rajah Tupas invites PC to come to his home and watch the eclipse. Some tribe members approach Rajah Tupas to report that the ruler of Alpas is present and his soldiers are causing a commotion in the tribe. Lord Sigmund XXXIX, the ruler of Alpas, is ordering the soldiers to search the

homes of every tribe member and find PC. PC runs and hides.

After the soldiers have left, PC goes out of hiding and returns to Rajah Tupas. The festival food and decorations and their homes are in ruins. The soldiers' reckless raids injured some mythicas. PC then apologizes and explains to Rajah Tupas.

Rajah Tupas tasks PC to catch a Chalf, a tamaraw-like mythica, which is needed to continue the festivity. PC locates the rare mythica in the dungeon near the tribe. After fulfilling Tupas' task, he rejects PC's proposition to help Sulayman's plan. PC convinces the rajah again, explaining that the earlier incident is why they should open the minds of the people in Alpas and make them realize that mythicas are no threat to humans. Tupas wanted to avenge Mayari and the injured mythicas, and liked the idea of taking down the ruler of Alpas.

Tupas thanks PC for this plan and for doing his favors. He then stated that the rare mythica caught by PC is the one used as an offering to Mayari.

The festivity continued happily with the total solar eclipse happening, serving as the feast's end.

Arc 2 – Liberation

- **Tribe Pínátúbó**

PC travels cautiously towards Tribe Pínátúbó. They were aware that this tribe was near the Alpas borders, and there might be soldiers in the area. When PC arrived, they saw tribe members and tribe women. Going to the main grounds of the tribe, they see Ligaya arguing with Gat Lóntók, the tribe leader of Pinatubo. Ligaya argues with Lontok because they did not have any mythicas living with them. Lontok explains that they followed the rule of Alpas, banning all mythicas.

However, because their tribe is not as advanced as Alpas, there are still some mythicas roaming around their lands. Despite tormenting and threatening them, the mythicas stayed. PC wondered how they could convince Lontok not to harm the mythicas and let them live together in perfect harmony. Ligaya suggests that they should show Lontok how humans and mythicas can work together, just like in previous tribes.

PC and Ligaya search for mythicas around the area. They saw a group of mythicas who ran towards the dungeon near the tribe. PC and Ligaya entered the dungeon. Mythicas started attacking them. They fought back, and after winning, the mythicas became submissive. PC and Ligaya brought knocked-out mythicas to Lontok to prove that humans and mythicas can work together. The tribe members were shocked and raised their guards. PC and Ligaya explained to Lontok how mythicas are useful as sparring partners and helpful in labor. They both described the lifestyle of the last tribes. PC demonstrated how they and their mythica fight in combat. Lontok accepted the idea but thought of mythicas as enslaved creatures they could utilize to their liking. Lontok accepts PC's proposition, but PC had some second thoughts.

Ligaya offered to stay in tribe Pínátúbó and observe Lontok's next steps. PC will return to this tribe after they ask for help from Suláymán.

- **On the way to the next tribe**

While PC was walking towards the next tribe, a unit of Alpas soldiers spotted them. They attacked the PC, and they fought back. Adil was in shock and unable to move when he realized that PC had a mythica. PC knocked down two soldiers and was ready to attack Adil. Adil learns that PC had a mission concerning mythicas and Alpas.

Adil lets them go and advises them to be careful of Ribuc's unit searching for them. PC could not believe that an Alpas soldier would let them go but hurriedly ran away.

- **Tribe Kánláón**

PC arrives at tribe Kánláon while remaining cautious and alert from possible Alpas soldiers. PC observes that the tribe's people mistreat mythicas. Humans treat mythicas as low beings and boss them around. Dayang Bángkáyá, along with some mythicas escorting him, approaches PC. Bangkaya proudly tours PC to their tribe and explains their relationship with the mythicas. She also brags about how their tribe is notorious for having solid mythicas. PC confronts Bangkaya and questions her morals. The tribe leader got angry and aggressive. Bangkaya calls out her most prominent and most vigorous mythica, and orders it to attack the PC.

After defeating the mythica, PC tells Bangkaya that they will convey to the other tribe leaders about her defeat and abuses towards the mythicas. However, PC will remain silent if Bangkaya joins PC and the other tribe leaders in breaking the barriers of Alpas for humans and mythicas to live in harmony. Scared of being an embarrassment, Bangkaya agrees with the proposition. At this point, PC realizes that humans find it easy to abuse its power and dominance towards the mythicas. PC sailed towards the next tribe.

- **Tribe Lánáyá**

PC arrived at tribe Lánáyá and noticed that the tribe was only occupying half the space of their island.

PC asked around about the whereabouts of their tribe leader, and some men

said that she was on the other side. PC followed the directions given to them. PC reached the other side. The place was full of mythicas, and the only human present was the tribe leader, Datu Kálángitán. Kalangitan was checking whether the mythicas still had a source of food, shelter, and clean water. PC approaches Kalangitan and talks with her.

Kalangitan explains why they have divided their island. Their tribe believes that mythicas have the right to live on their own. Being stuck with humans is against their nature. Some humans may genuinely care for a mythica, but many only abuse and use them for their interests. PC tells Kalangitan about their mission. Kalangitan explains that harmony is excellent. However, humans should also consider whether the mythicas want to live with humans. Kalangitan asks PC to persuade at least ten mythicas to go to the humans' side of the island for a bonfire. Mythicas should not be forced and must willingly go to bond with humans. Kalangitan further explains that this mission might be complex due to the mythicas' horrible experiences with humans. The only human that the mythicas trust is her. If PC succeeds, Kalangitan will believe that mythicas want to harmonize with humans and accept PC's proposition. PC may persuade the mythicas by giving them food, bonding, and close combats. During this mission, PC gains more realizations about the mythicas' emotions. PC might be confused about their morals—PC sails towards the next tribe.

Arc 3 – Dilemma

- **Tribe Duláng**

PC meets Hidáet, mother of Ligaya, here and learns about family history. PC knows about the reasons why Sigmund and the explorers left Alpas. PC learns why

Suláymán wants to harmonize humans and mythicas, and why she wants to rule Alpas. However, Lumaya and her mythica remain a mystery to PC. Adil shows up with Ligaya. Adil helped Ligaya in saving and freeing the mythicas from Tribe Pínátúbó. Ribuc finally finds the PC. Adil helps PC.

Level Requirements

Level Diagram

Table 1: Level Diagram

Chapter	Level	Enemy level	Number of enemies
Act 1 – Harmony	Tutorial – Dream	Lvl 1	Low
	Introduction – Suláymán	Lvl 1	Low
		Lvl 2	Low
	Lvl 1 – Tribe Ópón	Lvl 2	Medium
		Lvl 5	Low
		Tribe Leader Battle	1
	Lvl 2 – Tribe Áltó	Lvl 5	Medium
		Lvl 7	Low
		Lvl 5 Dungeon Battle	High
	Lvl 3 – Tribe Íráyá	Lvl 7	Medium
		Lvl 10	Low
		Lvl 7 Dungeon Battle	High
		Rare Mythica Battle	1
Act 2 – Liberation	Lvl 1 – Tribe Pínátúbó	Lvl 10	High
		Lvl 15	Medium
		Lvl 10 Dungeon Battle	High
	Lvl 1.1 – Path to Tribe Kánláón	Lvl 20 Alpas Soldiers	Low
	Lvl 2 – Tribe Kánláón	Lvl 15	High
		Lvl 20	Medium
		Tribe Leader Battle	1
	Lvl 3 – Tribe Lánáyá	Lvl 20	High
		Lvl 25	Medium
Act 3 – Dilemma	Final Level – Tribe Duláng	Lvl 30	High
		Lvl 35	High

Asset Revelation Schedule

Table 2: Asset Revelation Schedule

Items	Recovers	Enhance/ Diminish	Duration
HP Recovery Items	+25%		
LL Recovery Items	+25%		
Base Exp Yield Boost		+50%	30 minutes
TR Reduction Items		-20%	30 seconds
Attack Boost		+20%	30 seconds
Defense Boost		+20%	30 seconds
Special Defense Boost		+15%	30 seconds

Level Design Seeds

Act 1 is Harmony with three levels and the Tutorial and Introduction. Dream should be the tutorial stage; here, Buwanawa gives PC the option to choose a starting Mythica and tutorials on the game's controls and basic combat. Suláymán will be the introduction stage, introducing the PC to the game's world and lore. PC should encounter levels 1 to 2 enemies that allow PC to gain 1 to 2 levels.

Tribe Ópón is the first level which will introduce PC to the Tribe Ópón and their way of living with their mythicas. PC should encounter levels 2 to 5 enemies and fight the Tribe Leader at the end of this level.

Level 2 is Tribe Áltó. PC will be again introduced to another way of mythicas living together with the tribe members. Unlike Tribe Ópón, who use their mythicas for combat, Tribe Áltó utilizes mythicas for agriculture. The tribe leader gives PC a task to retrieve java plum seeds in a nearby dungeon to finish this level. Enemies in this area are between levels 5 and 7, and many level 5 enemies populates the dungeon Level 3 is Tribe Íráyá. PC will arrive during the tribe's festival. PC's arrival will cause

some disturbance that will halt the festival temporarily. The Tribe Leader will instruct PC to locate and capture a legendary Mythica in a nearby dungeon. PC will encounter level 7 to 10 enemies around the area.

Just like in Act 1, Act 2 will also have three levels. In level 1, PC will encounter the Tribe Pínátúbó. Unlike other tribes, Tribe Pínátúbó agrees with Alpas on the ban of mythicas. However, since Tribe Pínátúbó does not have the same technology as Alpas, PC should still be able to encounter level 10 to 15 enemies in the area. The PC will have to capture mythicas in a nearby dungeon to finish the level. Level 10 enemies will populate the dungeon. PC will encounter and battle a few levels 20 Alpas Soldiers with medium difficulty along the path to the next level.

In level 2, PC will encounter the Tribe Kánláón, who mistreat their mythicas. PC will encounter level 15 to 20 enemies in the area. PC will have to battle and defeat the Tribe Leader to finish the level.

In level 3, the PC will encounter non-hostile (will not attack unless provoked) level 20 to 25 enemies. To finish this level, PC must accomplish the Tribe Leader's request to persuade mythicas to join the bonfire through social activities such as feeding them, bonding, or friendly battles.

Act 3 is the final act and the final level. The revelation of the whole history and lore of Pearl Isle to PC will occur. A great battle will occur, and the PC will encounter many levels of 30 to 35 enemies with hard difficulty.

CHAPTER III

Software Development and Testing

Development Environment and Tools

The project uses Unity, a cross-platform game engine developed by Unity Technologies. It allows the creation of games in 2D and 3D, and it offers a primary scripting API in C#. The researchers use Blender for the 3D assets and environment. Blender is a 3D computer graphics software that is free and open-source. The researchers also use Adobe Photoshop for UV editing, creating textures for 3D models created in Blender, normal maps, and marketing materials. The Assets created using both Blender and Photoshop will be imported to Unity. Unity will require an IDE to program the game. Visual Studio Community 2022 is an IDE with C# programming language, which Unity uses.

Pre-production

World Design

In a story-based RPG, it is essential to plan out the world. This way, the players would be able to interact with the world made by the developers in a way that they could understand how it works. Before doing any designs and development of the project, the researchers designed and conceptualized the in-game world so that they could refer to the initial conceptualization of this world during development.

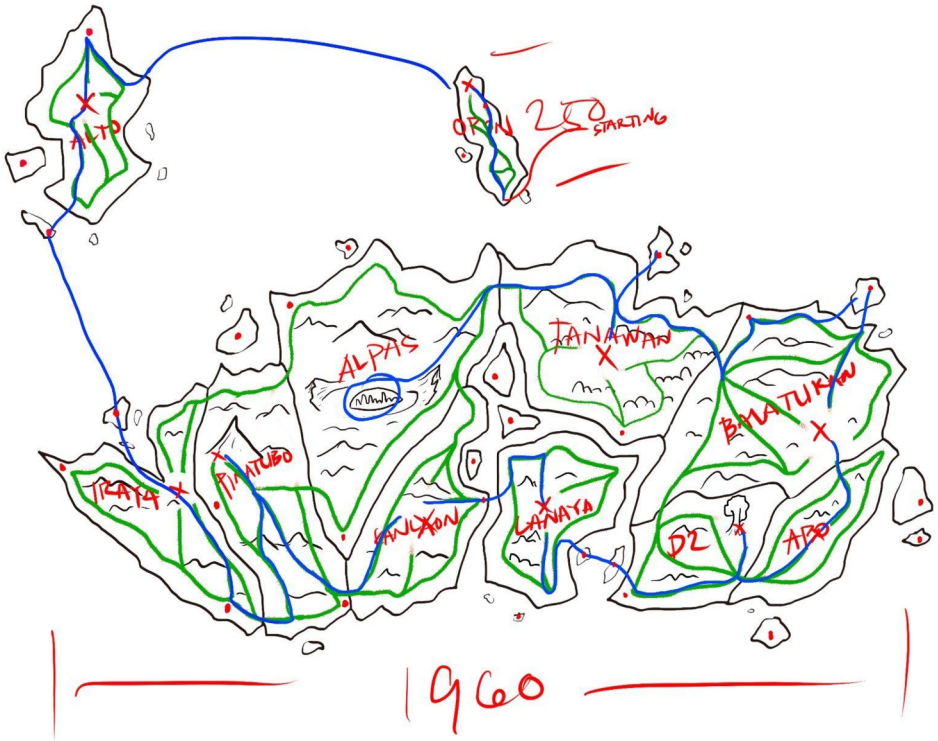


Figure 29: World Map Sketch
Blue lines are the intended player's path in the story. The green path is the road system that is established in the world. Red crosses represent the placement of the main grounds of the tribe. Red dots represent the dungeons located.

The following is the **world's backstory:**

Alpas was once the most developed tribe a millennia ago. It was the tribe where the first trade in Pearl Isles was made and where the tribe's people in the isle first discovered their ability to tame mythicas. The isle's current currency system, the barter system, was made possible by the tribe members there. It was a tribe ahead of its time. Alongside the prosperous tribe is the most powerful tribe leader known to all tribes as Rajah Isugáyá. She was part of the ancestry of powerful mythica tamers and was a respectable tribe leader of all of Pearl Isle. She was known to be strict with men as tribe members feared her and would respect her at all costs. Nobody would like to be her husband, and, in her perspective, she does not want weak and fearful men to be her

husband, too. Only those courageous enough to challenge her in a mythica battle are worthy.

One day, a group of foreign people visited the shores of Pearl Isle. As the hospitable tribe leader, Isugáyá would always greet foreigners like she always does. As she welcomes these foreigners, she notices the advancement of machinery they own and is astonished that devices like theirs exist in their time. She thinks the devices feel alien to her and would like to study their advancements in detail. The technologies the explorers have should be applied to the great tribe of Alpas as well. The explorer's captain was called Sigmund. Captain Sigmund was fond of Isugáyá as he taught her all the devices they brought to the land. Because of this, the captain extended his stay in the tribe. He created many types of machinery to make Alpas an even more prosperous tribe than before. Not long after, Isugáyá fell for Sigmund. Sigmund also fell for Isugáyá and soon later married, had kids, and reigned Alpas for many years. However, this was not the only reason the explorers ventured in the first place.

The explorers were a group of advanced pirates venturing around the globe searching for natives whose primary advances were driven by taming mythicas. Because of what they have learned throughout the years, they know that these advances never need the mythicas' help. As a result, the explorers vowed to never use mythicas as tools again, so they ventured. Teaching tribes worldwide how these advances could be used without the mythicas aids. However, Sigmund thought Alpas were different from the tribes they visited.

The captain thought that Alpas was destined for something greater. So, he established Alpas, and founded Alpas as the first city in Pearl Isle. The most advanced

and civilized people live there. Over the years, the foreigners in the city, including Captain Sigmund, made the people distance themselves from using the mythicas. Entirely changing the Alpas tribe's people's lifestyle altogether. They made mythicas banned in the once tribal city, even the word "mythica" should never be mentioned in the public's ears. The once respectable tribe leader, Isugáyá, and those from her ancestry were hated among all tribes in the isle.

Then, the tribe south of Alpas, Tribe Tán-áwán, became the capital tribe of all of Pearl Isle. Since Alpas isolated themselves from the whole isle, Tán-áwán automatically became the hotspot for trade. In that same tribe, fast forward to the present, 30 years before all the current events, there was an experienced mythica tamer. She was Lumáyá. Lumáyá was a people person, a jolly happy-go-lucky young girl destined to be a tribe leader. Together with her partner mythica, Separaghoul, she often greets people and helps them with their needs in the happy tribe of Tán-áwán. She was labeled "master" or "the great Lumáyá." She was even called "the present-day Isugáyá" by some. She was also often mistaken as "Rajah Lumáyá" by the people of Tribe Tán-áwán even if they had an existing tribe leader known as Rajah Suláymán. However, even for that, she still insists on being called only Lumáyá to pay respect to Suláymán.

One day, sick of the people anticipating Lumáyá as the tribe leader, Rajah Suláymán challenged Lumáyá to a mythica battle; however, Lumáyá refused the challenge since it was not the time for her to overtake Suláymán's throne. Pissed off by Lumáyá's comments, she commanded her partner mythica to attack Lumáyá's Separaghoul to death. Lumáyá was shocked by what she saw; she could not even move or even say a word. She has just seen her partner killed, and she could not

defend him. Right there and then, she decided to run away and never come back to the tribes ever again. Suláymán's ruthless kill of Lumáyá's partner made her become the most feared tribe leader in all of Pearl Isle.

The world backstory helps both in the gameplay design aspect and the game's character design. However, it is not the only material that could help create a diverse and real-world within the game. The researchers then set rules for the world to develop coherent rules for the gameplay design and the character design.

The game has two goal choice overall: to harmonize humans and mythicas and to liberate mythicas from the abuses of human-kind.

To achieve the harmonization of humans and mythicas, the PC should convince 8 tribe leaders, tame Buwanawa and defeat Lord Sigmund XXIX.

To achieve the liberation mythicas, the PC should still convince 8 tribe leaders. Along the way the PC will get to decide if they should tame Adlakawa and defeat Rajah Suláymán.

An optional path to take is to go to different dungeons to get a piece of the story in history.

The researchers have also set the naming origins of the tribe as part of the rules set in the world. Tribes will be named after the mountains and hills of the Philippines unless otherwise, reasons that may vary. The tribe names will be: Tribe Tán-áwán (Osmeña Peak), Tribe Ápó (Mt. Apo), Tribe Kánláon (Mt. Kanlaon), Tribe Pínátúbó (Mt. Pinatubo), Tribe D2 (Mt. Dulang-dulang), Tribe Íráyá (Mt. Iraya), Tribe Bálátúkán (Mt. Balatukan), Tribe Lánáyá (Mt. Lanaya), Tribe Opón, Tribe Áltó (Alto Peak).

Another rule set is that the modeled NPCs will be mixed upon the tribe. Ethnic groups the NPCs will be based on Aetas, Bisayas, Hiligaynon, and Sama Bajau.

The researchers have also set the naming origins of the tribe leaders. Tribe Leaders should be named after the Datus, Rajahs, Sultans, or Apos in the Philippines.

Gameplay Design

We plan to integrate these gameplay mechanics to fit within the game's RPG element while giving new experiences to the players. The project adds a few game mechanics from other genres, such as MOBAs, and open-world RPGs. It will allow the researchers to clearly understand the game's identity and what the game will become in the future.

Types of Mythicas

There are distinct mythica types in the game, each having different strengths and weaknesses. The table below shows the mythica's type advantages and disadvantages over the other types. The first row displays how the mythica's type affects the damage defensively, while the first column shows the mythica's type affects the damage offensively. Values above one show the type is at an advantage, while below one shows otherwise.

Table 3: Type Balancing Chart

DEFENSE → ATTACK ↴	PIERCER	BRAWLER	SLASHER	CHARGER	EMITTER	KEEPER
PIERCER	**1**	**0.75**	**1**	**1.5**	**1**	**0.75**
BRAWLER	**1**	**0.75**	**1**	**0.75**	**1**	**0.75**
SLASHER	**1.5**	**1**	**1.5**	**0.75**	**1.5**	**0.75**
CHARGER	**0.75**	**1.5**	**1.5**	**1**	**1.5**	**0.75**
EMITTER	**1.5**	**0.75**	**1.5**	**1.5**	**1.5**	**0.75**
KEEPER	**1**	**1**	**1**	**1**	**1**	**1.5**

There are six types of mythicas in the game, namely, **Piercer**, **Brawler**, **Slasher**, **Charger**, **Emitter**, and **Keeper**. The project's creators considered the mythica's type when designing a mythica specie visually.

Piercers have spear-like spikes and their designs follow that of spearmen. They are at an advantage when dealing with the Chargers. They can also resist attacks from Chargers. However, they are weak to Slashers and Emitters.

Another type of mythica is the Brawler type. They have tough skin and bodies. The heavy infantry and weapon shells are the inspiration for this mythica type. They might have strong body composition, but they have no mythica types that are an advantage. However, they are resistant to Piercers, Brawlers, and Emitters. Their rigid composition will not be strong enough to resist Chargers as they are weak against them.

Sworders are the main inspiration for Slashers. These mythicas are faster,

lighter, and have blade-like body features. However, they are less armed. The mythica type is strong to Piercers, Slashers, and Emitters. However, they are not resistant to other mythica types due to their less armed nature. Piercers and Emitters also have an advantage over these types of mythicas.

One of the characteristics of Chargers is their strong legs. The cavalry inspires its visual design. Brawlers, Slashers, and Emitters are weak against these mythicas. They are not resistant to any other mythica types. However, Slashers, Chargers, and Emitters have an advantage to this type of mythica.

Emitters are ranged, attackers. They emit projectiles and the archer is its main inspiration. They are firm to Piercers, Slashers, Chargers, and Emitters. Due to its advantage of being range, they are not resistant to other mythica types. However, Slashers, Chargers, and other Emitters could stand in the way of this mythica type.

Lastly, the Keeper mythicas are the most resistant mythicas. Most of the powerful mythicas are Keeper mythicas. Their armor is the hardest. They are guardians and have this sense of royalty. Only other Keepers could handle them, and they are resistant to all other mythica types.

Stats Calculation

The stats calculation refers to the following monster stats: **Health**, **Tame Resistance**, **Physical Attack**, **Physical Defense**, **Special Attack**, and **Special Defense**. The researchers chose the stat calculation in Pokémon as the main inspiration for this project's stat calculation. The said game serves as an inspiration for this project. A comprehensive discourse about the different stats is on pg. 11 of this study.

$$Stats = \left\lfloor \left(\frac{(Base + SV) \times Level}{MaxLevel} \right) + Level \right\rfloor$$

Equation 1: Stats Calculation

Damage Calculation

The calculation refers to calculating the amount of damage a mythica can hit. Like the stat calculation, Pokémon is also the main inspiration for this calculation. However, to fit the game's dynamic difficulty adjustment feature, the GAMEPACE variable in the formula is added.

$$Damage = (GAMEPACE \times Level) \times \left(\frac{Attack}{Defense} \right) \times \left(\frac{Power}{MaxPower} \right) \times Modifier$$

Equation 2: Damage Calculation

First, the **Damage** factor is the total damage output intended to deduct the opponent's HP.

Next is the **Level**, which describes a mythica attacker's level of experience.

The **GAMEPACE** refers to how much of the combat system in the game will be nerfed or boosted. It helps in changing the combat pace of the game, either making combat end faster or slower depending on the amount. The lesser the value, the slower the combat pace will be.

Attack refers to the mythica attacker's base attack. If the attacker's skill is a **Physical** Skill, then it would take the attacker's base physical attack stat. However, if the attacker's skill is a **Special** Skill, it would take the attacker's base special attack stat.

TotalDefense refers to the target's total base defense, including other factors such as held items. If the attacker's skill is a **Physical** Attack, it will take the target's base physical defense stat. However, if the attacker's skill is a **Special** Attack, it would

take the attacker's base special defense stat.

The **Power** calculation factor refers to the skill's base power. By default, It is at most 255 and at least 0.

MaxPower refers to the skill with the highest base power of all the skills. (255) Lastly, the Modifier has many factors to consider. Below is the Modifier's calculation.

$$Modifier = STAB \times \left(1 + \frac{SV}{100}\right) \times Type \times Critical \times Random$$

Equation 3: Modifier Calculation

The **Modifier** variable is part of the Damage Calculation formula. This variable will affect the overall damage based on the factors to consider: **STAB**, **SV**, **Type**, **Critical**, and **Random**.

The factor STAB means the *Same Type Attack Bonus*. From the game perspective, if the mythica attacker uses a skill similar to its type, it will have an attack bonus. The number will be one of the attackers uses a skill that is not the same as the mythica's type. 1.5 will be the STAB value.

The next factor is the SV or the **Stability Value**. Each mythica the player captures has a random stability value. This stability value makes each mythica the player encounters unique. Since the stability value is 0-50, it will be divided by 100 to get the percentage added to 1. The greater the stability value the player's mythica has, the higher the modifier will be.

Type refers to whether the attacker mythica's skill type is strong against or resisted the target's type. It depends on how the skill matches against the opposing target. The type refers to the type table on page 61. (i.e., if the attacker is strong against the opposing target, the Type value will be the value 1.5, if the opposing target resists

the attacker's skill type, then the type value will be the value .75).

Another one is **Critical**, which is a random hit bonus. The opposing target mythica will deal significant damage. 2 will be the Critical value if the *Random Number Generator* (RNG) decides a critical hit and one if the RNG decides otherwise. It depends on the mythica attacker's critical rate and other factors such as held items. By default, 5% will be the lowest possible critical rate.

Lastly, the factor **Random** refers to the attack's damage rolls. Most of the attacks in a game have a damage roll in which the attack will not have a fixed value every time the attacker uses a skill or attacks. The game generates a random number between 0.75 to 1. It means the modifier will reduce to 75% or stay the same. Note: Since critical hits, STAB, and SV exist, the maximum value of the damage roll should not exceed 1.

Tame Value Calculation

Total Tame Value

It is the calculation of the amount of Tame Value a wild mythica has for it to be tamed.

The researchers should note that the constant value of 200 in the formula means that the lowest possible value the Tame Value will get is 200. A level 1 mythica should get at least 200 Tame Value considering without Status Effect and Full HP.

$$TameValue = 200 + \left(\frac{Level^3}{5}\right) \times StatusFX \times \left(\frac{CurrentHP}{MaxHP}\right)$$

Equation 4: Tame Value Calculation at Full HP

The **Level** factor refers to the wild mythica's current level. **StatusFX** refers to if the mythica has a status effect present or not. If the mythica has a status effect, .75 will

be the value; one will be the value if otherwise. 25% will be taken off the Tame Value if the mythica has a status effect. **CurrentHP** refers to the wild mythica's current HP. **MaxHP** refers to the wild mythica's total HP value.

This calculation intends only when the percentage of the current HP of the mythica is equal to or greater than 50%. Below is the calculation of Tame Value if the mythica has below 50% current HP.

$$TameValue = 200 + \left(\frac{Level^3}{5}\right) \times StatusFX \times 0.5$$

Equation 5: Tame Value Calculation at 50% HP

Even at below 50% of the current HP, a mythica tame value deduction caps at 50%.

Tame Beam Value

The calculation of the Tame Beam, the player's Taming Staff, will accumulate to capture a mythica. Several monster taming games such as Temtem, Pokemon, and Kindred Fates are all inspirations in creating this unique formula.

$$TameBeam = AvgLevel^2 \times \left(\frac{TSPower}{TR}\right) \times Random \times GAMEPACE$$

Equation 6: Tame Beam Calculation

The factor **AvgLevel** refers to the total average level of all the player's mythica currently in the party.

TSPower refers to the Power the player's Taming Staff has. If the player upgrades their Taming Staff, it will increase its Power. (0-255)

The next factor, the **TR**, refers to the Taming Resistance of the mythica the player is targeting. So, certain species of mythica are more challenging to capture than

others.

The factor **Random** refers to the Tame Beam roll. It differentiates each Tame Beam the player gives to the wild mythica. The game gives a random number of 0.85 to 1 to the result. The tame beam value will be deducted to 85% or will stay the same.

Lastly, **GAMEPACE** refers to how much of the Taming system in the game will be nerfed or boosted. It helps in making the taming process easier. The lesser the value, the slower the tame pace will be.

Experience Calculation

The amount of experience an individual mythica has indicated how much it has battled and combat.

Experience Level Up Requirement Calculation

The calculation displayed below is the number of experience points required for a mythica to reach a certain level.

$$EXP = Level^3$$

Equation 7: Experience Points Calculation

The calculation's level factor refers to the next level the player's mythica would want to reach.

In order to explain the formula elaborately, the following situation serves as an example: If the player's mythica wants to reach level 20, it should accumulate 8,000 experience points to reach the said level. So, if the mythica is level 19, it should have at least 6,859 experience points and gain at least 1,141 experience points to reach level 20.

The basis of this formula is from *Pokémon's Medium Fast group* experience calculation which is cubic.

Experience Gain Calculation

The amount of experience that a mythica gives when defeated depends on its level and its species. The higher level of the defeated mythica, the more experience points it yields. However, numerous factors can influence how much experience any individual mythica gains.

$$EXPGain = \left\lfloor \frac{GAMEPACE \times Wild \times BaseEXP \times (1 + EXPBonus) \times Level \times Evolve \times Type \times Random}{(Mult - (GAMEPACE - 0.5)) \times Mythicas} \right\rfloor$$

Equation 8: Experience Gain Calculation

GAMEPACE refers to how much of the experience point system in the game will be nerfed or boosted. It helps in changing the difficulty of the entire game by either giving each mythica a bonus by increasing the value or making the game harder by giving the player's mythica fewer experience points than usual.

The **Wild** factor refers to if the defeated mythica is from the wild or if it is a Tamer battle. The value will be one if it is in the wild and 1.5 if otherwise. If a tamer owns the mythica defeated, it will give more experience points than the average mythica from the wild.

The **BaseEXP** is the base experience yield statistic of the fainted mythica's species.

EXPBonus is the amount of bonus added to the mythica if it holds an item that adds experience bonus points. For example, if a mythica holds a 50% experience point bonus, EXPBonus will be 0.5, adding this to 1, resulting in 1.5. It means that the player gains 150% of the original experience points.

A factor in the calculation is the **Level** factor which refers to the level of the defeated mythica.

Evolve refers if the winning mythica is at or past the level where it would be able to evolve, but it has not. 1.2 would be multiplied if it could evolve but not; one would be otherwise.

Another factor in the calculation is **Type**, which refers if the winning mythica is weak against the opponent mythica. If it is, then 1.2 would be multiplied, one if otherwise.

The **Random** factor refers to the experience roll. Each defeated mythica will have a distinct experience calculation every time. The system will roll a random number between .9-1.2 for the calculation. It means the entire calculation decreases to 90%, or it will increase to up to 120%

The **Mult** factor is complicated. It is the number of multipliers in the numerator of the equation. The value, 8, subtracts to the GAMEPACE value and subtracts to the GAMEPACE value's minimum. The subtraction occurs from the GAMEPACE value because the higher the GAMEPACE value, the greater the value deducted to Mult and the lesser the value of the denominator. This process gives the player more EXP and makes the game easier. The same goes otherwise: GAMEPACE value is subtracted to 0.5 because the minimum amount of the GAMEPACE value is 0.5, so if the value of GAMEPACE is 0.5 (meaning the player is experiencing the most challenging mode of the game), the Mult will still be equal to 8, making the denominator a much greater value, giving you fewer experience points, making the game even harder.

The **Mythica** factor refers to the number of mythica participating in the combat.

Barter System (Crafting)

The barter system is the world's "crafting" system. Bartering will be the main currency to capture the inspiration for the game (Filipino pre-colonial culture). The barter system replaces the in-game crafting system and currency system. However, both crafting and bartering will have the same mechanics and results. There is a fixed requirement for each item when bartering.

AI for Dynamic Difficulty Adjustment (AI for DDA)

AI for DDA is a method of automatically modifying a game's features, behaviors, and scenarios in real-time, depending on the player's skill. When the game is easy, the player does not feel bored or frustrated when it is exceedingly difficult.

Table 4: AI for Dynamic Difficulty Adjustment Parameters and Data to Collect

Parameters to Adjust:	**Data to collect:**
Symbol beside the parameter explains the value to increase the game's difficulty.	**The symbol beside the data explains what conditions should the data be to increase the game's difficulty.**
● ↓ **GAMEPACE** - Adjusts damage, experience gain, and tame beam value.	● ↓ Number of Failed Encounters ● ↑ Average Party's level from previous encounter and current encounter ● ↓ Number of failed attempts to tame/capture a mythica
● ↑ **Wild Mythica Level Multiplier** - All wild mythica spawners should refer to the wild mythica level multiplier and multiply the level values to increase the wild mythica's level.	● ↓ Number of Failed Encounters
● ↑ **Economy** – Item drop timer, barter requirements of an Item, and amount of item drops calculated with the Economy value to ensure to increase or decrease said the values.	● ↓ Number of Failed Encounters ● ↑ Average Party's level from previous encounter and current encounter

Items

Items are in-game equipment that provides mythicas with bonus attributes or consumables that could provide ease to tamers.

Most essential items can be purchased from the merchant, while others are dropped by objects in nature, wild mythicas, or even only found in the chests of ancient dungeons around the overworld.

Each mythica has six item slots. Some items provide bonus attributes that are stackable, while others do not.

Only consumable items are stackable or can have more than one amount in one slot, and Upgrade items should only have one in a slot.

Some items are losable, meaning if the player faints in combat, some items that mythicas have equipped are gone permanently.

Character Design

In making the character designs, we acquired references from sites like DeviantArt, Pinterest, and other games that have some similarities to the game's genre and theme. We first created the world and then populated the world. Using the created world, we can determine what kind of lifestyle, culture, and beliefs these characters would have in this world. We use that information to design characters that fit well into their environment, making them more realistic and immersive to players.

Figure 30: Player Characters Design: Eleo and Luna (Design Concept)

Figure 31: Mythica Design: Chanaque (Design Concept)

Figure 32: Unique NPC Design: Suláymán (Design Concept)

Environment Design

The Philippine Islands, specifically Cebu, Lapu-Lapu, and Leyte, inspire the game's world with significant changes to make it into a world that fits our planned theme and vision. We used Unity's terrain tool to create the world into a unique and prosperous environment that the players can immerse themselves in.

Figure 33: World Map Design

UI Design

The game’s UI design uses rounded quads and minimalistic design to achieve a kid-friendly look to the game. The game’s intended audience is the same as Nintendo Co., Ltd., which intends to deliver family-friendly games while not neglecting older players. This design philosophy would help invite the target audience to try the game themselves. The UI’s position is inspired by other games so that players who have played many games can immediately familiarize themselves with the UI.

Figure 34: Genshin Impact In-game Screen Capture

We positioned the party portraits on the right side of the screen, like Genshin Impact's UI. This intended position is for players to manage each of their team members' status easily, whether they are ready to be deployed for battle or not, if they do not have health, or which of the members are down or still alive. The location of the mini-map is in the upper left corner, which is a standard style for all video games.

Figure 35: League of Legends In-game Screen Capture

Another genre that we took inspiration from is the MOBA genre. Like in League of Legends, the item icons are shown on the lower right to make it easier for players to view the items they have equipped. It is located just below the party icons, for players to investigate just one corner of the screen to view everything mythica-related, like party status and items equipped.

Production

During the project's production, the creators have researched ways to export a significantly trivial build size. Because WebGL builds require small-sized exports, the project takes advantage of one of the main features of the game engine, Unity, called ScriptableObjects.

Unity's documentation page defines A ScriptableObject as a data container that the creators can use to save significant amounts of data, independent of class instances. One of the primary use cases for ScriptableObjects is to reduce the creator's project's memory usage by avoiding copies of values. It is helpful if the project has a Prefab that stores unchanging data in attached MonoBehaviour scripts.

Every time the system instantiates that Prefab, it will get its copy of that data. Instead of using the method and storing duplicated data, the creators can use a ScriptableObject to store the data and then access it by reference from all of the Prefabs. It means that there is one copy of the data in memory.

Like MonoBehaviours, ScriptableObjects derive from the base Unity object, but, unlike MonoBehaviours, the creators cannot attach a ScriptableObject to a GameObject. Instead, the creators need to save them as Assets in the project. When the creators use the Editor, they can save data to ScriptableObjects while editing

and at run time because ScriptableObjects use the Editor namespace and Editor scripting. However, in a deployed build, the creators cannot use ScriptableObjects to save data, but they can use the saved data from the ScriptableObject Assets they set up during development.

Data that the creators save from Editor Tools to ScriptableObjects as an asset is persistent between sessions because the system writes them on disk.

The ScriptableObject feature in Unity serves as the base feature used in systems made in the project. Systems that use the said feature include Distinct AI System (Pluggable AI); Monster System; Inventory System; Dialogue System; Sound System; and Decision-based Quest System. However, some scripts do not need the said data container feature. These scripts include Dynamic Difficulty Adjustment System; Sound System; the Core Player Scripts; User-interface (UI) Scripts; Combat System; Timeline Scripts (for Cutscenes); and Game Calculations.

Game Manager Script

The Game Manager script derives from MonoBehaviour class. It means it should be attached to a GameObject. The said script serves as the main initialization of the project's different managers, controllers, and handlers. A Singleton is a globally accessible class that exists in the game, but only once. The object that contains the Game Manager component will only have one instance within the project, and it is globally accessible to all scripts.

```
public class GameManager : MonoBehaviour
{

    public static GameManager instance;
    public DatabaseManager databaseManager;
    public UIManager uiManager;
    public AudioManager audioManager;
    public GameSceneManager gameSceneManager;
    public ObjectPooler pooler;
    public ScenePicker gameplayScene;
    public DynamicDifficultyAdjustment difficultyManager;
    public QuestManager questManager;
    public SaveManager saveManager;
    public PauseManager pauseManager;
    public PlayerInputHandler inputHandler;
    public TimelineManager timelineManager;
    public StateController gameStateController;
    public List<Transform> enemiesSeePlayer;

    [Header("Game States")]
    public State gameplayState;
    public State UIState;
    public State dialogueState;

    [HideInInspector] public string currentWorldScenePath = string.Empty;
    [HideInInspector] public Camera currentWorldCamera;
    [HideInInspector] public Player.Player player;
    [HideInInspector] public PlayerSaveData loadedSaveData;
```

Figure 36: Game Manager: Initialization of Managers, Controllers, and Handlers

The first part of the Game Manager script is a serialization of all managers, controllers, and handlers. Some fields cache virtual objects such as the player, the current World Scene the player is in, the current Camera used, and the Save Data loaded.

```
void Awake()
{
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(instance.gameObject);
    }
    else
    {
        Destroy(instance.gameObject);
        instance = this;
        DontDestroyOnLoad(instance.gameObject);
    }

    databaseManager.InitializeTypeChartData();
}
```

Figure 37: Game Manager: Awake

On the Awake method, this is the initialization of the Game Manager Singleton and the system calculation of the Type Chart Data.

```
public void InitializePlayerReference(Player.Player p)
{
    player = p;
}

public void InitializeCurrentWorldCamera(Camera cam)
{
    currentWorldCamera = cam;
}
```

Figure 38: Game Manager: Initialization of Player and Current World Camera

Some methods initialize the Player object and the current World Camera used. The former occurs whenever the Gameplay scene initializes, and the latter occurs whenever a different camera in the world opens.

```
public void DifficultyUpdateAdd(string dataKey, float valueToAdd)
{
    var data = difficultyManager.GetDataNeeded(dataKey);
    var dataValue = data.value;
    data.ChangeValue(dataValue + valueToAdd);
    difficultyManager.DataAdjusted(dataKey);
}

public void DifficultyUpdateChange(string dataKey, float newValue)
{
    var data = difficultyManager.GetDataNeeded(dataKey);
    data.ChangeValue(newValue);
    difficultyManager.DataAdjusted(dataKey);
}
```

Figure 39: Game Manager: Dynamic Difficulty Adjustment Updates

The calling of difficulty updates occurs in the Game Manager as the calling of these methods happens whenever there are added data (or subtracted depending if the value passed is a negative or positive float) or data that completely changes its value.

```
public void UpdateEnemiesSeePlayer(Object enemyTransform, out int enemyCount)
{
    enemyCount = enemiesSeePlayer.Count;
    for (var i = 0; i < enemyCount; i++)
    {
        var enemy = enemiesSeePlayer[i];
        if (enemyTransform != enemy) continue;
        enemiesSeePlayer.Remove(enemy);
        enemyCount--;
        break;
    }
}
```

Figure 40: Game Manager: Updating Enemies that See Player

There is also a method to update the enemiesSeePlayer list within the Game Manager instance. It occurs by checking if the passed object is an enemy or not, and if it is not, the instance removes it from the enemiesSeePlayer list.

Player Script

The main Player object has this component. Since the initialization of the Gameplay scene is once, there shall only be one Player object. This player object initializes in the Game Manager, so scripts that need the main Player object will have a reference for it.

```
[RequireComponent(typeof(StateController))]
public class Player : MonoBehaviour, IEntity, IHaveMonsters, IHaveHealth, ICanTame
{
    [Foldout("Data", true)]
    [ReadOnly] public string playerName;
     public List<MonsterSlot> monsterSlots;
    [ReadOnly] public EntityHealth playerHealth;
    [ReadOnly] public PlayerInventory playerInventory;
    public Dictionary<string, Monster> discoveredMonsters = new Dictionary<string, Monster>();
```

Figure 41: Player: Data

A Foldout attribute, a feature in the MyBox library that groups variables in the Unity Inspector, labeled “Data” that contains the player’s name, the player’s current party (monsterSlots), the player’s current health, the player’s inventory, and a dictionary of discovered monsters contains the first part of the Player script. The caching of these occurs on initialization or whenever it is updated.

```
[Foldout("Settings", true)]
[DisplayInspector] public PlayerSettings playerSettings;
[ReadOnly] public ProjectileRelease projectileReleases;
[ReadOnly] public GameObject unitIndicator;
[ReadOnly] public GameObject vectorIndicator;
```

Figure 42: Player: Settings

A Foldout attribute labeled “Settings” contains different references to the player’s settings. The PlayerSettings class is a ScriptableObject containing all the Prefabs used during gameplay. ProjectileRelease is a class that contains the different Transform objects, where the player releases projectiles. The unit and the vector indicator are the

graphics for which the former contains the unit sprite, and the latter contains the vector skill indicator sprite.

```
[HideInInspector] public float tempSpeed;
[HideInInspector] public float tempAttackRate;
[HideInInspector] public SelectionManager selectionManager;
[HideInInspector] public Camera mainCamera;
[HideInInspector] public GameObject tamer;
private Health _healthComponent;
[HideInInspector] public SkillManager skillManager;
[HideInInspector] private PlayerInputHandler _inputHandler;
[HideInInspector] public Rigidbody rgdbody;
[HideInInspector] public Animator currentAnimator;
[HideInInspector] public PlayerQuestManager playerQuestManager;
private StateController _stateController;
[HideInInspector] public MonsterManager monsterManager;
[HideInInspector] public MonsterSlot monsterAttacker;
[HideInInspector] public PlayerSaveData savedData;
[HideInInspector] public Vector3 colliderExtents;
private Transform _playerTransform;
private readonly Vector3 _zeroVector = Vector3.zero;
private DateTime _dateOpened;
[SerializeField] private bool _tamerInvulnerable = false;
private readonly Color _white = Color.white;
```

Figure 43: Player: Hidden Fields

The above variables are cached components of the Player object, fields needed for some components and scripts within the game, and serialized components and sprites.

```
void Awake()
{
    savedData = GameManager.instance.loadedSaveData;
    GameManager.instance.uiManager.generalOptionsUi.ChangeUIValues();

    Init();
    TransferPlayerPositionRotation(savedData.playerWorldPlacement);
}

#region Initialization

private void Init()
{
    if (GameManager.instance == null) return;
    _playerTransform = transform;
    GetNeededComponents();
    InitializePlayerSavedData();
    monsterManager.ActivateMonsterManager(this, skillManager);
    tempSpeed = playerSettings.playerData.speed;
    tempAttackRate = playerSettings.playerData.attackRate;
    unitIndicator.transform.localScale = new Vector3(playerSettings.tameRadius, playerSettings.tameRadius, playerSettings.tameRadius);
    _stateController.ActivateAI(true, null, this);
    GameManager.instance.uiManager.InitGameplayUI(playerName, playerHealth.currentHealth, playerHealth.maxHealth, monsterSlots);
    GameManager.instance.uiManager.loadingScreen.gameObject.SetActive(false);
}
private void GetNeededComponents()
{
    mainCamera = GameManager.instance.currentWorldCamera;
    skillManager = GetComponent<SkillManager>();
    skillManager.skillSlots.Clear();
    monsterManager = GetComponent<MonsterManager>();
    selectionManager = GetComponent<SelectionManager>();
    selectionManager.ActivateSelectionManager(this);
    _inputHandler = GameManager.instance.inputHandler;
    rgdbody = GetComponent<Rigidbody>();
    _inputHandler.ActivatePlayerInputHandler(this, mainCamera);
    _inputHandler.EnterGameplay();
    _stateController = GetComponent<StateController>();
    _healthComponent = GetComponent<Health>();
    playerQuestManager = GetComponent<PlayerQuestManager>();
    colliderExtents = GetComponent<Collider>().bounds.extents;
}
```

Figure 44: Player: Awake, Init, GetNeededComponents

The above figure shows the Awake method, which calls the Init (Initialization) method, gets the saved data from the Game Manager, changes the value of the options UI based on the saved data, and transfers the player's position based on the saved data. The Init method gets the needed components of the player, Initializes the player data, activates the Monster Manager, stores the default player speed, stores the default attack rate, changes the unit indicator to its tame radius, sets the cursor to the default cursor, activates the player's state controller, initializes the GameplayUI based on the saved data, and deactivates the loading screen UI.

```
private void InitializePlayerSavedData()
{
    if (savedData == null) return;

    playerName = savedData.name;
    monsterSlots = savedData.playerMonsters;
    playerHealth = savedData.playerHealth;
    playerInventory.inventorySlots = savedData.inventorySlots;
    playerQuestManager.activeQuests = savedData.activeQuests;
    GameManager.instance.uiManager.questUI.UpdateQuestIcons(savedData.activeQuests.Values.ToList());
    playerQuestManager.finishedQuests = savedData.finishedQuests;
    discoveredMonsters = savedData.discoveredMonsters;
    var playerGFX = Instantiate(savedData.sex.Equals(Sex.Male) ? playerSettings.male : playerSettings.female, _playerTransform);
    tamer = playerGFX;
    SetPlayerSavedData();
}
private void SetPlayerSavedData()
{
    //after getting all data,
    var monsterAvgLvl = GameSettings.MonstersAvgLevel(monsterSlots);
    tamer.layer = LayerMask.NameToLayer("Player");
    currentAnimator = tamer.GetComponent<Animator>();

    //initialize player's health
    if (GameSettings.MonstersAvgHealth(monsterSlots.ToList()) <= 0)
    {
        playerHealth.maxHealth = GameManager.instance.saveManager.defaultPlayerHealth;
    }
    else
    {
        playerHealth.maxHealth = GameSettings.Stats(
            GameSettings.MonstersAvgHealth(monsterSlots.ToList()),
            GameSettings.MonstersAvgStabilityValue(monsterSlots.ToList()),
            monsterAvgLvl);
    }

    _healthComponent.UpdateHealth(playerHealth.maxHealth, playerHealth.currentHealth);
    //initialize party's avg level for difficulty adjustment
    GameManager.instance.DifficultyUpdateChange("Average Party Level", monsterAvgLvl);
    _dateOpened = DateTime.Now;
}
```

The above figure shows the initialization of the saved data, and from the saved data, the program calculates the different needed stats of both the player and the monsters.

```
public PlayerSaveData GetCurrentSaveData()
{
    try
    {
        savedData = new PlayerSaveData(savedData.name,
            savedData.sex,
            new WorldPlacementData(_playerTransform.position, _playerTransform.rotation,
                _playerTransform.localScale),
            monsterSlots,
            playerHealth,
            playerInventory.inventorySlots,
            GameManager.instance.currentWorldScenePath,
            discoveredMonsters,
            playerQuestManager.activeQuests,
            playerQuestManager.finishedQuests,
            savedData.timeSpent + (DateTime.Now - _dateOpened),
            DateTime.Now,
            GameManager.instance.uiManager.generalOptionsUi.GetCurrentOptionsData());
    }
    catch
    {
        //ignored
    }

    return savedData;
}
```

Figure 46: Player: Get Current Save Data

The above figure shows the method that returns a PlayerSaveData object that directly gets the player's data values. It gets called when the calling of the Save method in the Save Manager occurs.

```
public int GetCurrentSlotNumber()
{
    return _inputHandler.currentMonster;
}

public float GetMonsterSwitchRate()
{
    return playerSettings.playerData.monsterSwitchRate;
}

public bool SwitchMonster(int currentSlot, out string message)
{
    if (currentSlot < 0)
    {
        monsterManager.SwitchToTamer();
    }
    else
    {
        if (monsterSlots[currentSlot].fainted)
        {
            message = "Monster selected already fainted.";
            return false;
        }
        monsterManager.SwitchMonster(currentSlot);
    }

    message = string.Empty;
    return true;
}

public List<Monster> GetMonsters()
{
    var monsters = new List<Monster>();
    if (monsterSlots.Count <= 0) return monsters;
    monsters.AddRange(monsterSlots.Select(slot => slot.monster != null ? slot.monster : null));
    return monsters;
}
```

Figure 47: Player: Get Current Monster, Get Monster Switch Rate, Switch Monster, Get Current Monsters List

The figure above serves as the interface members' starting point (the Player component derives from IHaveMonsters). The interface has the following members: GetCurrentSlotNumber, which returns the current slot number the tamer or monster is in; GetMonsterSwitchRate, which returns the switch rate in which a tamer or the player

can switch monsters to; SwitchMonster, which handles the switching of an IHaveMonster object to a specific monster or back to its original tamer form; and GetMonsters returns a list of monsters of the IHaveMonsters object.

```
public void AddNewMonsterSlotToParty(int slotNum, MonsterSlot newSlot)
{
    monsterSlots[slotNum] = newSlot;
    monsterSlots[slotNum].slotNumber = slotNum;
    monsterSlots[slotNum].inParty = true;
    monsterManager.RequestPoolMonstersPrefab();
    monsterManager.GetMonsterAnimators();
    GameManager.instance.uiManager.UpdatePartyUI(monsterSlots[slotNum]);
}

public void AddNewMonsterSlotToStorage(MonsterSlot newSlot, out int slotNum)
{
    slotNum = 0;
    var storageMonstersCount = storageMonsters.Count;
    for (var i = 0; i < storageMonstersCount; i++)
    {
        if (storageMonsters[i].monster != null) continue;

        slotNum = i;
        break;
    }

    storageMonsters[slotNum] = newSlot;
    storageMonsters[slotNum].slotNumber = slotNum;
    storageMonsters[slotNum].inParty = false;
}
```

Figure 48: Player: Add New Monster Slot to Party, Add New Monster Slot to Storage

Above shows methods for adding a new monster to the party and storage. The calling of these methods occurs whenever the Player tames a new monster. Adding the monster to the party is called when the party is not complete and to the storage when the party is not whole.

```
public List<MonsterSlot> GetMonsterSlots()
{
    return monsterSlots;
}

public MonsterSlot GetMonsterWithHighestExp()
{
    var mSlot = new MonsterSlot();
    foreach (var slot in monsterSlots.Where(slot => mSlot.monster == null || mSlot.currentExp >= slot.currentExp))
    {
        mSlot = slot;
    }
    return mSlot;
}

public Monster GetCurrentMonster()
{
    return _inputHandler.currentMonster < 0 ? null : monsterSlots[_inputHandler.currentMonster].monster;
}

public GameObject GetTamer()
{
    return tamer;
}
```

Figure 49: Player: Get the Monster Slots (Party), Get Monster with Highest Experience Points (Party), Get the Current Monster Switched, Get the Tamer

The figure above shows a method of returning the Party-list of monster slots, which returns the monster with the highest experience points in the party, returns the currently switched monster, and returns the tamer.

```
public void ChangeStatsToMonster(int slot)
{
    tempSpeed = playerSettings.playerData.speed;
    tempAttackRate = playerSettings.playerData.attackRate;

    if (slot < 0)
    {
        playerHealth.maxHealth = GameSettings.Stats(
            GameSettings.MonstersAvgHealth(monsterSlots.ToList()),
            GameSettings.MonstersAvgStabilityValue(monsterSlots.ToList()),
            GameSettings.MonstersAvgLevel(monsterSlots.ToList())
        );
        _healthComponent.UpdateHealth(playerHealth.maxHealth, playerHealth.currentHealth);
        return;
    }

    tempSpeed *= monsterSlots[slot].monster.stats.movementSpeed;
    tempAttackRate *= monsterSlots[slot].monster.stats.attackRate;

    //Initialize Monster's health
    var maxHealth =
        GameSettings.Stats(
            monsterSlots[slot].monster.stats.baseHealth,
            monsterSlots[slot].stabilityValue,
            GameSettings.Level(monsterSlots[slot].currentExp)
        );
    _healthComponent.UpdateHealth(maxHealth, monsterSlots[slot].currentHealth);
}
```

Figure 50: Player: Change Stats to Monster

The above figure shows a method to change the stats of the player wherein the calling of the said method occurs every time the player wants to switch to a new monster.

```
public void AddToDiscoveredMonsters(Monster monster)
{
    try
    {
        discoveredMonsters.Add(monster.ID, monster);
    }
    catch
    {
        //ignored
    }
}
```

Figure 51: Player: Add to Discovered Monsters

The above is the last method that involves Monsters. The calling of the said method occurs whenever the Player discovers a new monster.

```
public void ChangeMonsterUnitIndicatorRadius(float radius)
{
    unitIndicator.transform.localScale = new Vector3(radius, radius, radius);
}

public void ReleaseBasicAttack()
{
    var monAttacking = GetCurrentMonster();

    var range = monAttacking.basicAttackType != BasicAttackType.Melee;
    var rotation = _playerTransform.rotation;
    var projectile = GameManager.instance.pooler.
        SpawnFromPool(range ? null : projectileReleases.front, monAttacking.basicAttackObjects.projectile.name,
            monAttacking.basicAttackObjects.projectile, range ? projectileReleases.front.position : _zeroVector, range ? rotation : Quaternion.Euler(-90, rotation.y, rotation.z));

    var rangeProjectile = projectile.GetComponent<IDamageDetection>() ?? projectile.AddComponent<Projectile>();
    var target = selectionManager.selectables.Count > 0 ? selectionManager.selectables[0] : null;
    var deathTime = range ? .25f : .1f;
    var speed = range ? 30f : 20f;

    rangeProjectile.ProjectileData(true, range, monAttacking.basicAttackObjects.targetObject, monAttacking.basicAttackObjects.impact,
        monAttacking.basicAttackObjects.muzzle, false, true, _playerTransform, target,
        _zeroVector, deathTime, speed, .5f, monAttacking.basicAttackSkill);
}
```

Figure 52: Player: Change Monsters Unit Indicator Radius, Release Basic Attack

The above starts with the Player's methods that involve combat. Above are methods of changing the radius of the Unit Indicator and the release of the monster's basic attack. The calling of the latter method only occurs when the current tamer's form is a monster, not the tamer.

```
public void SpawnSwitchFX()
{
    GameManager.instance.pooler.
        SpawnFromPool(_playerTransform, playerSettings.tameBeam.projectileGraphics.targetObject.name,
            playerSettings.tameBeam.projectileGraphics.targetObject, _zeroVector,
            Quaternion.identity);
}

public void ReleaseTameBeam()
{
    if (_inputHandler.currentMonster >= 0) return;
    if (selectionManager.selectables.Count <= 0) return;

    var tameable = selectionManager.selectables[0].GetComponent<ITameable>();
    if (tameable == null)
    {
        Debug.Log("Target has to be a wild mythica.");
        return;
    }

    //spawn projectile
    var projectile = GameManager.instance.pooler.
        SpawnFromPool(null, playerSettings.tameBeam.projectileGraphics.projectile.name,
        playerSettings.tameBeam.projectileGraphics.projectile, projectileReleases.front.position,
        Quaternion.identity);
    var rangeProjectile = projectile.GetComponent<IDamageDetection>() ?? projectile.AddComponent<Projectile>();
    rangeProjectile.ProjectileData(true, true, playerSettings.tameBeam.projectileGraphics.targetObject, playerSettings.tameBeam.projectileGraphics.impact,
        playerSettings.tameBeam.projectileGraphics.muzzle, true, false, _playerTransform, selectionManager.selectables[0],
        _zeroVector, 10, 30, 1f, playerSettings.tameBeam.skill);
}
```

Figure 53: Player: Spawn Switch FX, Release Tame Beam

The above figure shows a method to spawn a Switch FX whenever the player switches to a new form, monster, or tamer. The calling of another method shown above occurs whenever the player's form is a tamer.

```
private void FindAliveMonsterOrPlayer()
{
    var monsterSlot = new MonsterSlot();
    var count = monsterSlots.Count;
    for (var i = 0; i < count; i++)
    {
        var monster = monsterSlots[i];
        if (monster.currentHealth > 0 && monster.monster != null)
        {
            monsterSlot = monster;
        }
    }

    if (monsterSlot.monster == null)
    {
        if (_inputHandler.currentMonster < 0) return;
        _inputHandler.currentMonster = -1;
        monsterManager.SwitchToTamer();
    }
    else
    {
        if (monsterManager == null) return;
        _inputHandler.currentMonster = monsterSlot.slotNumber;
        monsterManager.SwitchMonster(monsterSlot.slotNumber);
    }
}
```

Figure 54: Player: Find Alive Monster or Tamer

The method's calling shown above occurs whenever the player's current monster form faints during combat. The player will then automatically switch to another monster that has not fainted, or if no monster still has health, it will automatically change to the tamer.

```
public void AddExperience(int experienceToAdd, int slotNum)
{
    var nextLevel = GameSettings.Level(monsterSlots[slotNum].currentExp) + 1;
    var nextLevelExp = GameSettings.Experience(nextLevel);
    monsterSlots[slotNum].currentExp += experienceToAdd;

    if (monsterSlots[slotNum].currentExp > nextLevelExp)
    {
        //TODO: sound for leveling up
    }

    if (_inputHandler.currentMonster != slotNum) return;
    GameManager.instance.uiManager.UpdateExpUI(slotNum, experienceToAdd);
}
```

Figure 55: Player: Add Experience

The last method in the Player script that involves combat is to add experience points to the Player. The method calculates the monster's level that defeated another monster, adds a level, and adds experience points to the experience bar UI.

```
public void TakeDamage(int damageToTake)
{
    _healthComponent.ReduceHealth(damageToTake);
    var currentMonster = _inputHandler.currentMonster;

    if (currentMonster < 0)
    {
        playerHealth.currentHealth = _healthComponent.health.currentHealth;
        GameManager.instance.uiManager.UpdateHealthUI(currentMonster, playerHealth.currentHealth);

        if (playerHealth.currentHealth <= 0)
        {
            Die();
        }
        return;
    }
    monsterSlots[currentMonster].currentHealth = _healthComponent.health.currentHealth;
    GameManager.instance.uiManager.UpdateHealthUI(currentMonster, monsterSlots[currentMonster].currentHealth);

    if (monsterSlots[currentMonster].currentHealth > 0) return;
    monsterSlots[currentMonster].fainted = true;
    monsterSlots[currentMonster].currentLives--;
    FindAliveMonsterOrPlayer();
}
```

Figure 56: Player: Take Damage

The above figure shows the first method that involves health. This method and proceeding methods will be member methods of the interface, IHaveHealth. The calling of the said method occurs whenever the player takes damage. It analyzes if the player's current form is a monster or a tamer. If it is the tamer, the player dies, which resets the game to the last save state. If it is a monster, then this is where the calling of the FindAliveMonsterOrPlayer method occurs, wherein it finds the first monster that still has health and switches to it.

```
public void Heal(int amountToHeal)
{
    _healthComponent.AddHealth(amountToHeal);
    if (_inputHandler.currentMonster < 0)
    {
        playerHealth.currentHealth = _healthComponent.health.currentHealth;
        return;
    }
    monsterSlots[_inputHandler.currentMonster].currentHealth = _healthComponent.health.currentHealth;
}

public void RecordDamager(MonsterSlot slot)
{
    monsterAttacker = slot;
}
```

Figure 57: Player: Heal, Record Damager

The method, Heal, adds health to the current form of the Player. The method, Record Damager, caches the monster or entity that damaged the Player.

```
public void Die()
{
    if (_tamerInvulnerable) return;

    var playerGameObject = gameObject;
    tamer.SetActive(false);
    _inputHandler.movementInput = Vector2.zero;
    _inputHandler.activate = false;
    rgdbody.isKinematic = true;
    rgdbody.useGravity = false;
    GameManager.instance.pooler.SpawnFromPool(null, playerSettings.deathParticles.name, playerSettings.deathParticles, _playerTransform.position,
        Quaternion.identity);
    var playerOrigLayer = playerGameObject.layer;
    playerGameObject.layer = 0;
    skillManager.targeting = false;
    GameManager.instance.saveManager.activated = false;

    void Reset() => ResetGame(playerGameObject, playerOrigLayer);
    GameManager.instance.uiManager.modal.OpenModal("<color=#f48989>Game Over</color>", playerSettings.deathIcon, _white, Reset);
}
```

Figure 58: Player: Die

The calling of the method, Die, occurs whenever the current form of the Player is the tamer, and it has lost all its health. All components are reset and assign the Confirm button of the Modal UI to the ResetGame method, which resets the stats of the Player to the last save state.

```
private void FullyRestoreAllMonsters()
{
    var count = monsterSlots.Count;
    for (var i = 0; i < count; i++)
    {
        if (monsterSlots[i].monster == null) continue;
        var maxHealth = GameSettings.Stats(monsterSlots[i].monster.stats.baseHealth,
            monsterSlots[i].stabilityValue, GameSettings.Level(monsterSlots[i].currentExp));
        monsterSlots[i].currentHealth = maxHealth;
        monsterSlots[i].fainted = false;
        GameManager.instance.uiManager.UpdatePartyMemberHealth(i, maxHealth, maxHealth);
        for (var j = 0; j < monsterSlots[i].skillSlots.Length; j++)
        {
            if (monsterSlots[i].skillSlots[j] == null)
            {
                continue;
            }
            monsterSlots[i].skillSlots[j].cooldownTimer = 0;
            monsterSlots[i].skillSlots[j].skillState = SkillManager.SkillState.ready;
        }
    }
}

public void TickTamerInvulnerability()
{
    _tamerInvulnerable = !_tamerInvulnerable;
}
```

Figure 59: Player: Fully Restore All Monsters, Tick Tamer Invulnerability

The above figure shows the last methods that involve health. The method, FullyRestoreAllMonsters, restores the monsters to their total health, while the TickTamerInvulnerability method switches the _tamerInvulnerable value to true or false.

```
public void SetAnimator(Animator animatorToChange)
{
    currentAnimator = animatorToChange;
}

public Animator GetEntityAnimator()
{
    return currentAnimator;
}
```

Figure 60: Player: Set Animator, Get Entity Animator

The methods shown above are the only methods that involve the Animator component. The methods are only setting the current animator of the Player based on the Player's current form and returning the current animator.

```
public StateController GetStateController()
{
    return _stateController;
}
```

Figure 61: Player: Get State Controller

The figure shown above shows the only method that involves the StateController component. The method is only returning the current State Controller of the Player.

```
private void ResetGame(GameObject player, int layer)
{
    GameManager.instance.uiManager.loadingScreen.gameObject.SetActive(true);
    player.layer = layer;
    _inputHandler.activate = true;
    GameManager.instance.saveManager.activated = true;
    tamer.SetActive(true);
    rgdbody.isKinematic = false;
    rgdbody.useGravity = true;
    playerHealth.maxHealth = GameSettings.MonstersAvgHealth(monsterSlots.ToList()) <= 0 ?
        GameManager.instance.saveManager.defaultPlayerHealth :
        GameSettings.Stats(GameSettings.MonstersAvgHealth(monsterSlots.ToList()),
            GameSettings.MonstersAvgStabilityValue(monsterSlots.ToList()),
            GameSettings.MonstersAvgLevel(monsterSlots));

    playerHealth.currentHealth = playerHealth.maxHealth;
    _healthComponent.UpdateHealth(playerHealth.maxHealth, playerHealth.currentHealth);

    FullyRestoreAllMonsters();
    HandleItems(true);

    UnityAction resetPosition = () => TransferPlayerPositionRotation(savedData.playerWorldPlacement);
    UnityAction disableLoadingScreen = () => GameManager.instance.uiManager.loadingScreen.tweener.Disable();

    StopAllCoroutines();
    StartCoroutine(DelayAction(GameManager.instance.uiManager.loadingScreen.tweener.duration, resetPosition));
    StartCoroutine(DelayAction(2, disableLoadingScreen));
    GameManager.instance.uiManager.modal.CloseModal();
}
```

Figure 62: Player: Reset Game

The ResetGame method is part of the Miscellaneous portion of the Player script. The calling of the script only occurs when the clicking of the Confirm Button of the Modal UI happens, and the assignment happens on its onClicked event during the Die method of the Player script. The said method resets all of the Player's component settings, restores the health of both the tamer and the party, transfers the Player's position to the last save state, removes losable items during death, updates the UI, and closes the modal.

```
private void HandleItems(bool dead)
{
    if(!dead) return;

    var inventorySlotsCount = playerInventory.inventorySlots.Count;
    for (var i = 0; i < inventorySlotsCount; i++)
    {
        var slot = playerInventory.inventorySlots[i];
        if(slot.inventoryItem == null) continue;

        if (!slot.inventoryItem.losable) continue;

        slot.inventoryItem = null;
        slot.amountOfItems = 0;
    }

    var monstersCount = monsterSlots.Count;
    for (var i = 0; i < monstersCount; i++)
    {
        var itemsCount = monsterSlots[i].inventory.Length;
        if(monsterSlots[i].monster == null) continue;
        for (var j = 0; j < itemsCount; j++)
        {
            var slot = monsterSlots[i].inventory[j];
            if(slot == null) continue;
            if(slot.inventoryItem == null) continue;
            if(!slot.inventoryItem.losable) continue;

            slot.inventoryItem = null;
            slot.amountOfItems = 0;
        }
    }
}
```

Figure 63: Player: Handle Items (during Death)

The calling of the method shown above occurs whenever the player needs to remove losable items. One example of a situation where the player calls this method is when it dies, where the player loses all losable items.

```
IEnumerator DelayAction(float delay, UnityAction action)
{
    yield return new WaitForSeconds(delay);
    action?.Invoke();
}

private void TransferPlayerPositionRotation(WorldPlacementData placementData)
{
    if(placementData == null) return;

    _playerTransform.SetPositionAndRotation(placementData.position, placementData.rotation);
}
public bool SamePositionFromSaved()
{
    try
    {
        return savedData.playerWorldPlacement.position.Approximately(_playerTransform.position);
    }
    catch
    {
        return false;
    }
}
```

Figure 64: Player: Delay Action, Transfer Player Position Rotation, Same Position From Saved

The figure shown above displays the last methods of the Player script and the last methods of the Miscellaneous portion of the Player script. The DelayAction method is a Coroutine, and its calling occurs whenever the Player needs to delay action. The TransferPlayerPositionRotation method is called every time the Player transfers to a specific location. The SamePositionFromSaved method is called every time the Save Manager checks if the Player's position is in the same position as the last saved state.

Decision-based Quest System and Dialogue System

The quest system and dialogue system have their connection. In summary, dialogues with choices can contain quests or gives quests to the player depending on their choice during the dialogue. With this in mind, the explanation of the quest and dialogue systems in the project is further below.

```
[CreateAssetMenu(menuName = "Quest System/New Quest")]
public class Quest : ScriptableObjectWithID
{
    public string title;
    [TextArea(5,10)]
    public string description;
    public List<Reward> rewards;
    public QuestGoal goal;

    [Space] [Tooltip("Conversation to proceed if ever the current quest succeeds")]
    public Conversation successConversation;

    [Tooltip("Conversation to proceed if ever the current quest fails")]
    public Conversation failConversation;
}
```

Figure 65: Decision-based Quest System: Quest

The above figure shows the Quest class, which derives from the ScriptableObjectWithID class, a custom class derived from the ScriptableObject class built in the game engine. The ScriptableObjectWithID class is similar to the ScriptableObject class, and its only difference is that it automatically creates a Globally Unique Identifier (GUID) and stores it in the id string variable.

As said, ScriptableObjects are data containers, so the Quest class contains the following data: the title of the quest; its description; a list of Reward to be given to the player when succeeded; the quest's goal; and a list of Conversation to be given to the player if they succeed; or fails the said quest.

```
[System.Serializable]
public class Reward
{
    public QuestRewardsType rewardsType;
    public int value;
}

[System.Serializable]
public class PlayerAcceptedQuest
{
    public Quest quest;
    public Character questGiver;
    public int currentAmount;
    public bool completed;
    public DateTime dateAccepted;

    public PlayerAcceptedQuest(Quest newQuest, Character questGiver, DateTime dateAccepted)
    {
        quest = newQuest;
        this.questGiver = questGiver;
        currentAmount = 0;
        this.dateAccepted = dateAccepted;
        completed = false;
    }
}

public enum RewardTypesEnum
{
    Items, Experience,
}
```

Figure 66: Decision-based Quest System: Reward, Accepted Quest, RewardTypesEnum

The figure above shows the Reward class, which contains the Reward type and its value. Reward type could be an item or experience.

Another class displayed in the above figure is the PlayerAcceptedQuest class, in which the conversion of the Quests that the player accepts occurs into the class where it stores the Quest, the giver of the Quest, the current amount of the Quest, a boolean to signify if it has completed, and the date it was accepted.

Lastly, the above figure also shows an enum of the name RewardTypesEnum which contains the values: Items; and Experience.

```
public class QuestRewardsType
{
    public RewardTypesEnum typeEnumOfReward;
    [ConditionalField(nameof(typeEnumOfReward), false, RewardTypesEnum.Items)]
    public ItemObject rewardItem;
}
```

Figure 67: Decision-based Quest System: Quest Rewards Type

The above figure displays the QuestRewardsType class, which contains the type of reward, an item object if the type of reward is an Item.

```
public abstract class QuestGoal : ScriptableObject
{
    public int requiredAmount = 5;

    public virtual void Initialization(){}

    public bool IsComplete(int currentAmount)
    {
        return currentAmount >= requiredAmount;
    }
}
```

Figure 68: Decision-based Quest System: Quest Goal

The figure above shows the abstract class QuestGoal, deriving from the ScriptableObject class. The data container contains the required amount of the goal and two methods, namely Initialization and IsComplete method, that requires a pass of the current amount to compare it to the required amount.

```
public class PlayerQuestManager : MonoBehaviour
{
    public Dictionary<string, PlayerAcceptedQuest> activeQuests = new Dictionary<string, PlayerAcceptedQuest>();
    public Dictionary<string, PlayerAcceptedQuest> finishedQuests = new Dictionary<string, PlayerAcceptedQuest>();
```

Figure 69: Decision-based Quest System: Player Quest Manager

The above figure shows the PlayerQuestManager that contains two dictionaries called activeQuests and finishedQuests, which is a storage of all quests that the player accepts. Since this class derives from a MonoBehaviour class, this will be attached to the Player object in the scene.

```
public void GiveQuestToPlayer(Quest questGiven, Character questGiver)
{
    GameManager.instance.uiManager.questUI.UpdateQuestIcons(activeQuests.Values.ToList());
    if (PlayerHaveQuest(activeQuests, questGiven)) return;

    var newQuest = new PlayerAcceptedQuest(questGiven, questGiver, DateTime.Now);
    activeQuests.Add(newQuest.quest.ID, newQuest);
    GameManager.instance.audioManager.PlaySFX("Confirmation");
    GameManager.instance.uiManager.questUI.UpdateQuestIcons(activeQuests.Values.ToList());
}

public Dictionary<string, PlayerAcceptedQuest> GetTotalQuests()
{
    if (activeQuests != null && finishedQuests != null)
    {
        return activeQuests.Concat(finishedQuests.Where(x => !activeQuests.ContainsKey(x.Key))). //merge all quests where no duplicate keys
            OrderBy(x => x.Value.dateAccepted). //order by date accepted
            ToDictionary(x => x.Key, x => x.Value); //convert to dictionary
    }
    else if (activeQuests != null && finishedQuests == null)
    {
        return activeQuests;
    }
    else if (activeQuests == null && finishedQuests != null)
    {
        return finishedQuests;
    }

    return null;
}
```

Figure 70: Decision-based Quest System: Give Quest to Player, Get the Total Quests

The above figure displays two methods in the PlayerQuestManager class, namely GiveQuestToPlayer, which requires a given quest, and the giver of the quest, GetTotalQuests, returns all quests as a concatenation of the activeQuests and finishedQuests dictionary.

```
public bool PlayerHaveQuest(Dictionary<string, PlayerAcceptedQuest> questList, Quest quest)
{
    return questList != null && questList.ContainsKey(quest.ID);
}

public void RemoveQuestToPlayerAcceptedQuest(Quest questToRemove)
{
    activeQuests.Remove(questToRemove.ID);
    GameManager.instance.uiManager.questUI.UpdateQuestIcons(activeQuests.Values.ToList());
}
```

Figure 71: Decision-based Quest System: Player Have Quest, Remove Quest to Player's Quest

The above displays two methods in which PlayerHaveQuest returns true if a quest list, a dictionary of type PlayerAcceptedQuest, contains the Quest to be searched. The other method RemoveQuestToPlayerAcceptedQuest requires removing a specific quest to the activeQuests dictionary.

```
public void GetQuestRewards(PlayerAcceptedQuest acceptedQuest)
{
    var rewardsCount = acceptedQuest.quest.rewards.Count;
    for (var i = 0; i < rewardsCount; i++)
    {
        var questReward = acceptedQuest.quest.rewards[i];
        switch (questReward.rewardsType.typeEnumOfReward)
        {
            case RewardTypesEnum.Items:
                var item = questReward.rewardsType.rewardItem;
                var value = questReward.value;
                GameManager.instance.player.playerInventory.AddItemInPlayerInventory(item, value);

                if (item is Gold)
                {
                    GameManager.instance.uiManager.UpdateGoldUI();
                }
                break;
            case RewardTypesEnum.Experience:
                GameManager.instance.player.monsterManager.AddMonstersExp(questReward.value);
                break;
        }
    }

    try
    {
        finishedQuests.Add(acceptedQuest.quest.ID, acceptedQuest);
    }
    catch
    {
        //ignored
    }
}
```

Figure 72: Decision-based Quest System: Get Quest Rewards

The method's calling shown above occurs whenever the player has finished the quest. From the acceptedQuest, a loop to analyze each reward its reward type is an item or experience. If it is an item, then add the item to the supposed inventory of the player. If it is an experience reward, then the total experience to be added will be shared with all of the mythicas in the party.

```
public abstract class QuestGoal : ScriptableObject
{
    public int requiredAmount = 5;

    public virtual void Initialization(){}

    public bool IsComplete(int currentAmount)
    {
        return currentAmount >= requiredAmount;
    }
}
```

Figure 73: Decision-based Quest System: Quest Goal

A base class shown above called QuestGoal inherits from ScriptableObject, which contains an integer of requiredAmount, a virtual method called Initialization, and a method that returns a boolean called IsComplete, which determines if the goal is complete.

```
public class QuestManager : MonoBehaviour
{
    public void UpdateKillQuest(Monster monster)
    {
        var playerQuests = GameManager.instance.player.playerQuestManager.activeQuests.Values.ToList();
        var questCount = playerQuests.Count;

        for (var i = 0; i < questCount; i++)
        {
            var active = playerQuests[i];
            if (!(active.quest.goal is KillGoal killGoal)) continue;

            killGoal.EnemyKilled(active, monster, out var updated);
            active.currentAmount = updated;

            //if the kill goal is complete and active.completed isn't set to true
            if (killGoal.IsComplete(updated) && !active.completed) OnComplete(active);
        }
    }
```

Figure 74: Decision-based Quest System: Quest Manager, Update Kill Quest

Shown above is a MonoBehaviour class called QuestManager. The caching of Quest Manager occurs in the Game Manager script. The Quest Manager's job is to inform the Player Quest Manager that the achievement of a new Quest Goal occurred. It then goes through all the active quests the player has and checks if what was the completed goal, and updates it. It will then update the active quest's current amount and check if the current amount surpasses the required amount. It will update the completed boolean variable in the active quests class if it does. Shown above is the Update Kill Quest Method.

```
public void UpdateGatherQuest()
{
    var playerQuests = GameManager.instance.player.playerQuestManager.activeQuests.Values.ToList();
    var questCount = playerQuests.Count;

    for (var i = 0; i < questCount; i++)
    {
        var active = playerQuests[i];
        if (!(active.quest.goal is GatherGoal gatherGoal)) continue;

        gatherGoal.ItemGathered(active, out var updated);

        //if the gather goal is not complete or active quest is completed
        if (!gatherGoal.IsComplete(updated) || active.completed)
        {
            active.completed = gatherGoal.IsComplete(updated);
            continue;
        }

        OnComplete(active);
    }
}

private static void OnComplete(PlayerAcceptedQuest active)
{
    active.completed = true;
    GameManager.instance.uiManager.questUI.UpdateQuestIcons();
    GameManager.instance.audioManager.PlaySFX("Confirmation");
}
```

Figure 75: Decision-based Quest System: Update Gather Quest, On Complete

The method shown above is called Update Gather Quest, which is similar to the Update Kill Quest method. The only difference is that it checks which method is a gather goal and notifies the active quest that there is a gathered item. It will then update the

active quest's current amount and check if the current amount surpasses the required amount.

The calling of the OnComplete method occurs when the quest is complete.

Dialogue System

As mentioned earlier, the creators designed the dialogue system to have an option for assigning the players quests. Below are figures that show the Dialogue System and its connection to the Quest System.

```
[CreateAssetMenu(menuName = "Dialogue System/Character")]
public class Character : ScriptableObjectWithID
{
    public string fullName;
    public Sprite facePicture;
    public List<CharacterMood> moods;

    [SerializeField, HideInInspector] private bool _hasBeenInitialized = false;
    public Sex sexOfCharacter;
    public float dialoguePitch = 1f;

    [Conditional("UNITY_EDITOR")]
    private void OnValidate()
    {
        if(_hasBeenInitialized) return;

        var emotionMemberCount = Enum.GetNames(typeof(Emotion)).Length;

        for (var i = 0; i < emotionMemberCount; i++)
        {
            var moodToAdd = new CharacterMood(Enum.GetName(typeof(Emotion), i), (Emotion)i, null);
            moods.Add(moodToAdd);
        }

        _hasBeenInitialized = true;
    }
}
```

Figure 76: Dialogue System: Character

Shown above is the Character class, which inherits from the ScriptableObjectWithID class, which also inherits from the ScriptableObject class native in Unity. The container contains the character's full name, a sprite which is the face picture of the character, a list of CharacterMood, the sex of the character, and the pitch of the character's voice.

The OnValidate method applies when editing in the Unity Editor and does not apply in the build.

```
[Serializable]
public class CharacterMood
{
    public string name;
    public Emotion emotion;
    public Sprite graphic;

    public CharacterMood(string name, Emotion emotion, Sprite graphic)
    {
        this.name = name;
        this.emotion = emotion;
        this.graphic = graphic;
    }
}
```

Figure 77: Dialogue System: Character Mood

The CharacterMood class is a serializable class that contains the name of the mood, the motion, and the graphic of the mood. The class also contains a constructor to get the reference of the moods.

```
public interface IInteractable
{
    void Interact(Player player);
}
```

Figure 78: Dialogue System: IInteractable

The system also contains the IInteractable interface, which can inherit to interactable objects in the overworld.

```
[CreateAssetMenu(menuName = "Dialogue System/Conversation")]
public class Conversation : ScriptableObjectWithID
{
    public Line[] lines;
    [Tooltip("Displays with the last line of the nextConversation. Note: Please limit responseChoices to at most 5.")]
    public Choice[] choices;
}
```

Figure 79: Dialogue System: Conversation

The class shown above, called Conversation, is a ScriptableObject that contains an array of Lines and an array of type Choice. The dialogue system shows the choices at the end of the conversation.

```
[System.Serializable]
public struct Line
{
    [TextArea(2, 5)]
    public string text;
    public Character character;
    public Emotion emotion;
    public SpeakerDirection speakerDirection;
}

[System.Serializable]
public struct Choice
{
    public string text;
    public Conversation nextConversation;
    public Quest quest;
    public UnityEvent onClickChoice;
    [Foldout("Tooltip", true)]
    public string tooltipTitle;
    [TextArea(3,5)]
    public string tooltipDescription;
}
```

```
public enum SpeakerDirection
{
    Left,
    Right
}

public enum Emotion
{
    Normal,
    Happy,
    Sad,
    Angry,
    Surprised,
    Confused,
    Scared,
    Aggrivated
}
```

Figure 80: Dialogue System: Line, Choice, Speaker Direction, Emotion

The figure shown above displays four blocks of code. One would be a structure called Line, which contains the text of the dialogue line, the character, the character's emotion, and the speaker's direction in the Dialogue UI. Another would be a structure called Choice. It contains the text of dialogue choice, the following conversation after choosing the dialogue choice, an event to invoke when the player chooses the said dialogue choice, the tooltip title, and a description if we want a tooltip to display when hovering over the said dialogue choice. The last two blocks of code are enums called Speaker Direction, Emotion which lists constants of Left and Right, and constants emotions, respectively.

Dialogues will be called upon in the UI Manager since Dialogues are UI Elements in the game. The figures below are methods in the Dialogue UI class referenced in the

UIManager. Before anything else, the researchers need to define the Text Juicer, an open-source Unity plugin that allows for "per character animation" on text fields.

```
void Update()
{
    if (_dialogueTextJuicer.IsPlaying)
    {
        if(!nextLineIconTweener.isActiveAndEnabled) return;

        nextLineIconTweener.Disable();
        return;
    }
    CompleteTextJuicer();
    if (_nextLine == null)
    {
        _nextLine = nextLineIconTweener.gameObject;
    }
    _nextLine.SetActive(true);
}
```

Figure 81: Dialogue System: Dialogue UI: Update

The Update method, called every frame, checks if the _dialogueTextJuicer is playing and disables the next line icon if it is and returns. It runs the CompleteJuicer method and activates the icon object if it is not playing.

```
public bool TextJuicerPlaying()
{
    return _dialogueTextJuicer.IsPlaying;
}

public void CompleteTextJuicer()
{
    _dialogueTextJuicer.Stop();
    _dialogueTextJuicer.SetProgress(1f);
    _dialogueTextJuicer.enabled = false;
}
```

Figure 82: Dialogue System: Dialogue UI: Text Juicer Playing, Complete Text Juicer

The calling of the methods shown above occurs the TextJuicerPlaying, which returns a boolean that tells if the text juicer is playing; CompleteTextJuicer, which stops the text juicer, completes it, and disables the text juicer script.

```
public void StartDialogue(Conversation conversationToDisplay)
{
    var noChar = false;
    cutscene = false;
    _dialogueTextJuicer.SetDirty();

    //if its a new nextConversation
    if (_currentConversation != conversationToDisplay)
    {
        _lineCount = 0;
        _currentConversation = conversationToDisplay;
    }

    //if it's the last lineToDisplay in the nextConversation
    if (_lineCount == conversationToDisplay.lines.Length - 1)
    {
        EndConversation(conversationToDisplay.choices);
    }

    if (_speakerHolder == null)
    {
        _speakerHolder = speakerHolder.gameObject;
    }

    _speakerHolder.SetActive(false);
    nameHolder.SetActive(false);
    nameText.text = string.Empty;

    _currentCharacter = conversationToDisplay.lines[_lineCount].character;

    if (_currentCharacter == null)
    {
        _currentCharacter = GameManager.instance.loadedSaveData.sex == Sex.Male ? _maleCharacter : _femaleCharacter;
        noChar = true;
    }

    //change name box if its not the same value already
    if (!nameText.text.Equals(_currentCharacter.fullName))
    {
        nameText.text = noChar ? GameManager.instance.player.playerName : _currentCharacter.fullName;
    }
```

Figure 83: Dialogue System: Dialogue UI: Start Dialogue 1/3

The method shown above is the StartDialogue method, the primary method to call when a player interacts with an NPC containing a conversation. It needs a Conversation called conversationToDisplay. In this method, the first parts are initializing variables and setting the line count to zero when it is a different conversation than the last one. It will also check if the line count is at last and end the conversation by displaying the choices if it has one or end the dialogue if it has none. It will also

deactivate the _speakerHolder GameObject, the _nameHolder GameObject and set the dialogue's text to empty. It will then get the current character object, and if it has no character in the line of the conversation, it means that the player is the one talking. If it is, then it will get the sex of the player from the save file and gets the male or female Character referenced in the current class. It will then set the name text to the player's name or replaces the text with the full name of the character object if it has one.

```
    nameHolder.SetActive(true);
    //get the mood graphic of the character and initialize speaker picture placement
    var characterMoodGraphic = GetEmotionGraphic(_currentCharacter, conversationToDisplay.lines[_lineCount].emotion);
    InitializeSpeakerPicture(characterMoodGraphic, conversationToDisplay.lines[_lineCount].speakerDirection);

    _speakerHolder.SetActive(true);

    //change dialogue text
    ManageDialogueString(conversationToDisplay.lines[_lineCount].text);

    _lineCount++;

    SetDialogueUiTextJuicer();
}
```

Figure 84: Dialogue System: Dialogue UI: Start Dialogue 2/3

Shown above is the continuation of the Start Dialogue method. The method shows to enable the nameHolder GameObject, Initialize the speaker's picture, activate the _speakerHolder GameObject, Manage the Dialogue text, change the line count to the next line, set the Text Juicer.

```
public void StartDialogue(Line line, Choice[] choices, bool displayCharPic)
{
```

Figure 85: Dialogue System: Dialogue UI: Start Dialogue 3/3

The StartDialogue method also has a method of the same name that takes in a line, choices and a boolean that checks if the dialogue should display a character picture.

```
private void ManageDialogueString(string lineToDisplay)
{
    //change player tag
    if (GameManager.instance.loadedSaveData != null)
    {
        lineToDisplay = lineToDisplay.Replace(_playerNameTag, GameManager.instance.loadedSaveData.name);
    }

    //change mythica tag to first party mythica
    if (GameManager.instance.player != null)
    {
        try
        {
            var nickName = GameManager.instance.player.monsterSlots[0].name != string.Empty
                ? GameManager.instance.player.monsterSlots[0].name
                : GameManager.instance.player.monsterSlots[0].monster.monsterName;
            lineToDisplay = lineToDisplay.Replace(_firstPartyMythicaTag, nickName);
        }
        catch
        {
            //ignore
        }
    }


    dialogueText.text = lineToDisplay;
}
```

Figure 86: Dialogue System: Dialogue UI: Manage Dialogue String

The method above shows the managing of the dialogue string. Here, the method analyzes the text to see if it has a player tag, and if it has, then the text will be replaced with the player's name. It will also analyze if there is a mythica tag, and if it has, the replacement of the tag to the mythica's nickname occurs, and if it has none, the replacement of the tag to the mythica species' name occurs instead. Then, the system will change the dialogue text to the line to display.

```
private Sprite GetEmotionGraphic(Character character, Emotion lineEmotion)
{
    foreach (var mood in character.moods.Where(mood => lineEmotion == mood.emotion))
    {
        return mood.graphic;
    }

    return character.moods[0].graphic;
}
```

Figure 87: Dialogue System: Dialogue UI: Get Emotion Graphic

The method shown above is a method that returns the sprite of the mood.

```
private void InitializeSpeakerPicture(Sprite graphic, SpeakerDirection direction)
{
    //change speaker picture
    speaker.sprite = graphic;

    //switch graphic
    switch (direction)
    {
        case SpeakerDirection.Left:
            if (speakerHolder.anchoredPosition.x > 0)
            {
                speakerHolder.SetPositionX(speakerHolder.anchoredPosition.x * -1f);
            }
            if (speakerHolder.localScale.x < 0)
            {
                var tempScale = speakerHolder.localScale;
                tempScale.x *= -1f;
                speakerHolder.localScale = tempScale;
            }

            if (choiceHolder.anchoredPosition.x < 0)
            {
                choiceHolder.SetPositionX(choiceHolder.anchoredPosition.x * -1f);
            }
            break;

        case SpeakerDirection.Right:
            if (speakerHolder.anchoredPosition.x < 0)
            {
                speakerHolder.SetPositionX(speakerHolder.anchoredPosition.x * -1f);
            }
            if (speakerHolder.localScale.x > 0)
            {
                var tempScale = speakerHolder.localScale;
                tempScale.x *= -1f;
                speakerHolder.localScale = tempScale;
            }
            if (choiceHolder.anchoredPosition.x > 0)
            {
                choiceHolder.SetPositionX(choiceHolder.anchoredPosition.x * -1f);
            }
            break;
    }
}
```

Figure 88: Dialogue System: Dialogue UI: Initialize Speaker Picture

The method shown above is a method that checks if the speaker's direction is left or right and places it to the left of the screen if it is, and places it to the right side if it says right.

```
private void SetDialogueUiTextJuicer()
{
    if (_thisObject == null)
    {
        _thisObject = gameObject;
    }
    _thisObject.SetActive(false);
    _thisObject.SetActive(true);

    _dialogueTextJuicer.enabled = true;
    _dialogueTextJuicer.SetProgress(0f);
    _dialogueTextJuicer.Play();

    StopAllCoroutines();
    StartCoroutine(PlayDialogueSound());
}
```

Figure 89: Dialogue System: Dialogue UI: Set Dialogue UI Text Juicer

The figure shown above is to set the Text Juicer. It is enabled, set its progress to zero, and set to play. A Coroutine will start to play the dialogue sound.

```
public bool CurrentConversationHasChoice()
{
    if (_currentConversation != null)
    {
        return _currentConversation.choices.Length > 0;
    }

    if (_choicesGameObject == null)
    {
        _choicesGameObject = choiceHolder.gameObject;
    }

    return _choicesGameObject.activeInHierarchy;
}

public bool IsEnd()
{
    try
    {
        return _lineCount >= _currentConversation.lines.Length;
    }
    catch
    {
        return true;
    }
}
```

Figure 90: Dialogue System: Dialogue UI: Current Conversation Has Choice, Is End

The methods shown above are the CurrentConversationHasChoice and the IsEnd, which return booleans if the current conversation has a choice, and the dialogue is at the end of the lines, respectively.

```
public void OnDialogueEnd()
{
    GameManager.instance.timelineManager.ResumeTimelineForDialogue();
    _currentConversation = null;
    if (mainDialogueTweener.disabled) return;
    mainDialogueTweener.Disable();
}

public void ContinueExistingDialogue()
{
    StartDialogue(_currentConversation);
}
```

Figure 91: Dialogue System: Dialogue UI: On Dialogue End, Continue Existing Dialogue

There are two methods shown above called OnDialogueEnd and ContinueExistingDialogue, which resets the dialogue UI at the end and continue the existing dialogue if it is not at its end, respectively.

```
private void EndConversation(Choice[] choices)
{
    var choiceLength = choices.Length;

    //if no responseChoices end
    if (choiceLength <= 0)
    {
        if (_newQuestGiven == null) return;

        GameManager.instance.uiManager.questUI.OpenPanelFromIcon(_newQuestGiven, _newQuestGiven.questGiver.facePicture);
        _newQuestGiven = null;
        return;
    }

    //initializing all choice for choice buttons
    for (var i = 0; i < choiceButtons.Length; i++)
    {
        if (!_dialogueChoices.TryGetValue(choiceButtons[i], out var choiceButtonGameObject))
        {
            choiceButtonGameObject = choiceButtons[i].gameObject;
            _dialogueChoices.Add(choiceButtons[i], choiceButtonGameObject);
        }
        //set objects to false in case it is already opened
        choiceButtonGameObject.SetActive(false);
        choiceButtons[i].tooltipTrigger.enabled = false;
```

Figure 92: Dialogue System: Dialogue UI: End Conversation 1/3

Shown above is a method called EndConversation, which takes in an array of choices. The method starts by checking if there are any choices in the conversation, and if it has none, it will check if there is a new quest given. It will return if it has none. If it has one, the panel will open to signal the player that there is a new quest, then ends.

If there are choices, then a loop is established. The loop will go through all the choices. The loop will start by initializing and adding the choice button to a dictionary to serialize it.

```
    //initialize only when there is an instance of responseChoices[choiceNum]
    try
    {
        var choiceNum = i;
        UnityAction buttonFunc = () => AddChoiceButtonFunction(choices[choiceNum]);

        //the the action to the botton as event and set the text of the button
        choiceButtons[i].SetTextActionOnClickButton(buttonFunc, choices[i].text);

        //enable object so it can be seen
        choiceButtonGameObject.SetActive(true);

        //if the choice does not entail adding a quest to the player, then set the title and content of the tooltip to the desired title and content on the choice
        choiceButtons[i].tooltipTrigger.enabled = true;
        if (choices[i].quest != null)
        {
            choiceButtons[i].tooltipTrigger.SetTitleContent(choices[i].quest.title, choices[i].quest.description);
        }
        else
        {
            choiceButtons[i].tooltipTrigger.SetTitleContent(choices[i].tooltipTitle, choices[i].tooltipDescription);
        }
    }
    catch
    {
        // ignored
    }
}
```

Figure 93: Dialogue System: Dialogue UI: End Conversation 2/3

The above figure shows the continuation of the EndConversation method. It gives a choice a button functionality on click, activates the choice button, then checks if the choice does not entail adding a quest to the player, then sets the title and content of the tooltip to the desired title and content on the choice, if it entails a quest, then set the title and content of the tooltip to the title and description of the quest. Then, the loop continues. The try-catch clause here checks if choiceButtons[i] is null, and if it is, then the loop continues without adding the button functionality to the choice.

```
    //activate choicePanel
    if (_choicesGameObject == null)
    {
        _choicesGameObject = choiceTweener.gameObject;
    }
    _choicesGameObject.SetActive(true);
}
```

Figure 94: Dialogue System: Dialogue UI: End Conversation 3/3

At the end of the method, the method will activate the choice panel GameObject.

```
private void AddChoiceButtonFunction(Choice choice)
{
    if (_choicesGameObject == null)
    {
        _choicesGameObject = choiceTweener.gameObject;
    }
    _choicesGameObject.SetActive(false);

    _newQuestGiven = GameManager.instance.player.playerQuestManager.GiveQuestToPlayer(choice.quest, _currentCharacter);

    choice.onClickChoice?.Invoke();

    if (choice.nextConversation != null)
    {
        StartDialogue(choice.nextConversation);
    }
    else
    {
        mainDialogueTweener.Disable();

        _lineCount = 0;
        _currentConversation = null;
        if (!cutscene)
        {
            GameManager.instance.inputHandler.EnterGameplay();
            if (_newQuestGiven != null)
            {
                GameManager.instance.uiManager.questUI.OpenPanelFromIcon(_newQuestGiven, _newQuestGiven.questGiver.facePicture);
                _newQuestGiven = null;
            }
        }
    }

    GameManager.instance.uiManager.tooltip.tooltipTweener.Disable();
}
```

Figure 95: Dialogue System: Dialogue UI: Add Choice Button Function

The method above is the choice button functionality to the choice. It starts by disabling the choice GameObject. It then gets the new quest given by the PlayerQuestManager. The invocation of all events in the choice will then occur. The method then asks if there is another conversation in the choice. If it has, then the dialogue will start again. If it has none, then the conversation ends.

```
public IEnumerator PlayDialogueSound()
{
    var text = dialogueText.text;
    var index = 0;
    while (_dialogueTextJuicer.IsPlaying)
    {
        try
        {
            if (!Char.IsPunctuation(text[index]) && !Char.IsWhiteSpace(text[index]) && _currentCharacter != null)
            {
                var dialogueSFXName = _currentCharacter.sexOfCharacter == Sex.Male
                    ? text[index] + "_MALE".ToUpperInvariant()
                    : text[index] + "_FEMALE".ToUpperInvariant();
                GameManager.instance.audioManager.PlaySFX(dialogueSFXName, _currentCharacter.dialoguePitch);
            }
        }
        catch
        {
            //ignored
        }

        yield return new WaitForSecondsRealtime(_dialogueTextJuicer.Delay * 2.5f);
        index++;
    }
}
```

Figure 96: Dialogue System: Dialogue UI: Play Dialogue Sound

The method shown above is a Coroutine call of the name PlayDialogueSound. The calling of it occurs when the line is on screen. This method will only play the dialogue sound every particular time, and the sound will depend on the current letter. The try-catch clause checks if the current index is not a letter and there are no letters in the text; then, an exception will be called but ignored.

Dynamic Difficulty Adjustment

Dynamic Difficulty Adjustment is one of the essential features of the project. It serves as the player's way to experience dynamic difficulty within the game. Below are figures that explain the Dynamic Difficulty Adjustment's methods and functions.

```
[System.Serializable]
public class DifficultyParameter
{
    public string name;
    public double value = 1f;
    public double minValue;
    public double maxValue;
    public double valueAdjustment;
    public Difficulty increaseDifficulty;
    public List<string> dataNeeded;

    private Dictionary<string, string> _dataNeeded = new Dictionary<string, string>();

    public bool HasData(string dataToSearch)
    {
        if (_dataNeeded.Count > 0) return _dataNeeded.ContainsKey(dataToSearch);

        var dataCount = dataNeeded.Count;
        for (var i = 0; i < dataCount; i++)
        {
            _dataNeeded.Add(dataNeeded[i].Replace(" ", string.Empty).ToLower(), dataNeeded[i]);
        }

        return _dataNeeded.ContainsKey(dataToSearch);
    }
```

Figure 97: Dynamic Difficulty Adjustment: Difficulty Parameter 1/2

The figure above shows the class Difficulty Parameter, which contains the name of the parameter, the value of the parameter, the minimum and maximum value of the parameter, the value adjustment of the parameter, an enum type Difficulty which analyzes what the value should be if we want to increase the difficulty and a list of data needed for analyzation. The method HasData is called whenever the system wants to check if the difficulty parameter has the data.

```
public void AdjustDifficultyParameterValue(Difficulty difficulty)
{
    //if we want the difficulty to increase
    if (difficulty == Difficulty.higher)
    {
        //check if what should be the parameter to make it more difficult
        switch (increaseDifficulty)
        {
            //if the parameter should be higher to make it difficult, then increase
            case Difficulty.higher:
                value += valueAdjustment;
                break;
            //if the parameter should be lower to make it difficult, then decrease
            case Difficulty.lower:
                value -= valueAdjustment;
                break;
        }
    }
    //if we want the difficulty to decrease
    else
    {
        //check if what should be the parameter to make it more difficult
        switch (increaseDifficulty)
        {
            //if the parameter should be higher to make it difficult, then decrease
            case Difficulty.higher:
                value -= valueAdjustment;
                break;
            //if the parameter should be lower to make it difficult, then increase
            case Difficulty.lower:
                value += valueAdjustment;
                break;
        }
    }

    value = Clamp(value, minValue, maxValue);
}
```

Figure 98: Dynamic Difficulty Adjustment: Adjust Difficulty Parameter Value

The figure above is a method to adjust the difficulty parameter value, and it takes into what difficulty the parameter should adjust. If the difficulty setting is "higher," the parameter should check if it should increase or decrease depending on the increaseDifficulty variable. If the parameter should be higher to increase difficulty, then the parameter's value will be added to the valueAdjustment variable. If the parameter should be lower to increase difficulty, a subtraction of the value occurs instead. If the difficulty should be lower, it will check if the parameter should be higher to increase difficulty then decrease the value, and vice versa.

```
public static double Clamp(double value, double min, double max )
{
    return (value < min) ? min : (value > max) ? max : value;
}
```

Figure 99: Dynamic Difficulty Adjustment: Clamp

A Clamp method is also introduced for the value not to be lower than the minimum amount or higher than the maximum amount.

```
[System.Serializable]
public class ParameterDataNeeded
{
    public string name;
    public float value;
    public float previousValue;

    public Difficulty increaseDifficulty;
    [ConditionalField(nameof(decreaseDifficultyOnEquals), true)]
    public bool increaseDifficultyOnEquals;

    [ConditionalField(nameof(increaseDifficultyOnEquals), true)]
    public bool decreaseDifficultyOnEquals;

    public bool IncreaseDifficulty()
    {
        if (increaseDifficultyOnEquals && previousValue.Approximately(value))
        {
            return true;
        }

        switch (increaseDifficulty)
        {
            case Difficulty.higher:
                return value > previousValue;
            case Difficulty.lower:
                return value < previousValue;
            default:
                return false;
        }
    }
```

Figure 100: Dynamic Difficulty Adjustment: Parameter Data Needed

The Parameter Data Needed class contains the name of the class, the current value of the data, and the previous value of the data. It also contains a Difficulty enum called increaseDifficulty, whose usage is to check if the parameter should be increased based on the previous and current value. The class contains a method called IncreaseDifficulty which analyzes the based on the value of data; a parameter should increase.

```
public bool DecreaseDifficulty()
{
    if (decreaseDifficultyOnEquals && previousValue.Approximately(value))
    {
        return true;
    }

    switch (increaseDifficulty)
    {
        case Difficulty.higher:
            return value < previousValue;
        case Difficulty.lower:
            return value > previousValue;
        default:
            return false;
    }
}

public void ChangeValue(float newValue)
{
    previousValue = value;
    value = newValue;
}
```

Figure 101: Dynamic Difficulty Adjustment: Decrease Difficulty, Change Value

Shown above is the DecreaseDifficulty method, which checks if a parameter should decrease. The ChangeValue method changes the value of the data needed for a parameter.

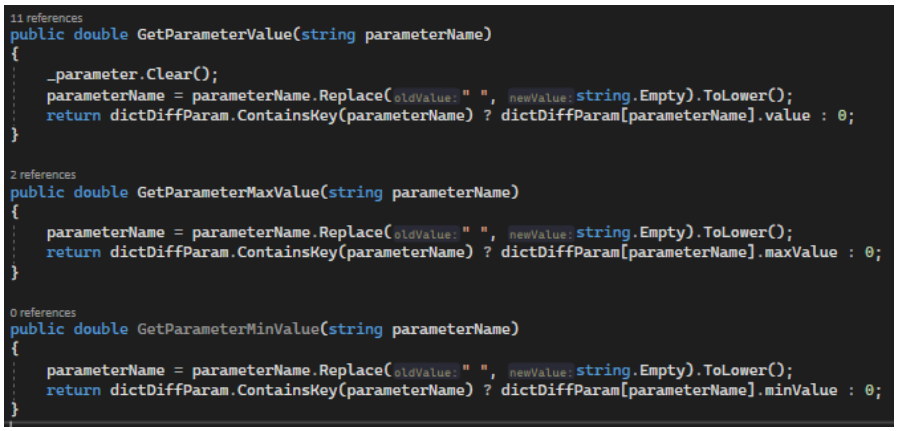
```
Unity Script | 1 reference
public class DynamicDifficultyAdjustment : MonoBehaviour
{
    public List<DifficultyParameter> difficultyParameters;
    public List<ParameterDataNeeded> parameterDataNeeded;

    private Dictionary<string, ParameterDataNeeded> dictParamDataNeeded = new Dictionary<string, ParameterDataNeeded>();
    private Dictionary<string, DifficultyParameter> dictDiffParam = new Dictionary<string, DifficultyParameter>();
    public HashSet<string> _parameter = new HashSet<string>();
    private TextMeshProUGUI ddaText;

    Unity Message | 0 references
    void Awake()
    {
        var paramCount :int = difficultyParameters.Count;
        var dataCount :int = parameterDataNeeded.Count;

        for (var i = 0; i < paramCount; i++)
        {
            dictDiffParam.Add(difficultyParameters[i].name.Replace(oldValue: " ", newValue: string.Empty).ToLower(), difficultyParameters[i]);
        }

        for (var i = 0; i < dataCount; i++)
        {
            dictParamDataNeeded.Add(parameterDataNeeded[i].name.Replace(oldValue: " ", newValue: string.Empty).ToLower(), parameterDataNeeded[i]);
        }
    }
```

Figure 102: Dynamic Difficulty Adjustment: Awake

Shown above is the DynamicDifficultyAdjustment class, which derives from a MonoBehaviour class. The GameManager cached this class, and accessing the said class is through it. The class contains a list of the DifficultyParameter objects and a list

of ParameterDataNeeded objects. It also contains a Dictionary of the parameters and the data needed for less memory consumption when searching.

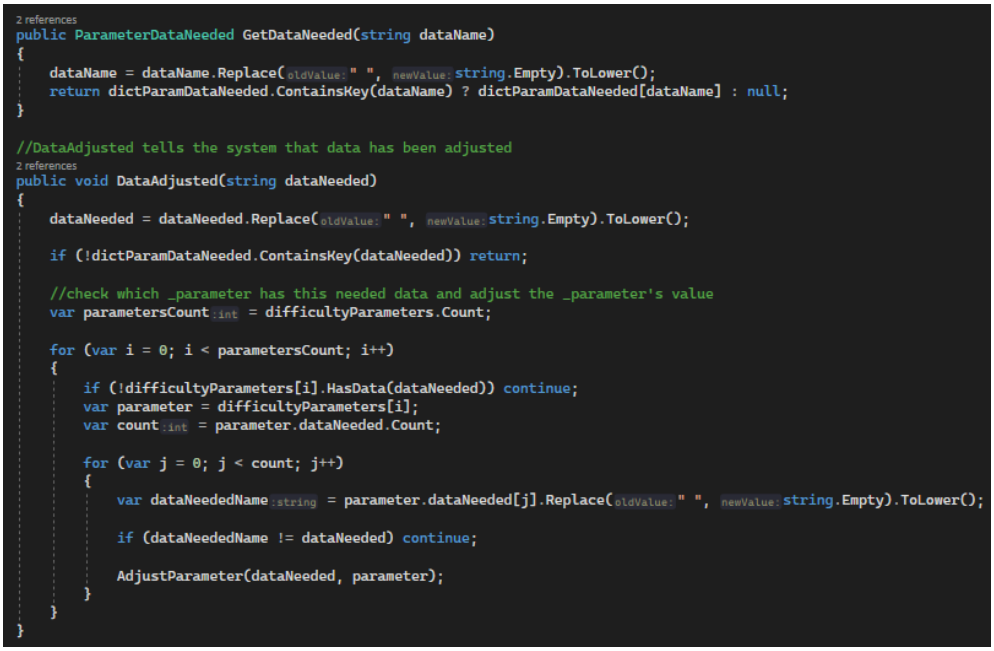
The Awake method adds the parameters and the data needed to the dictionary.



```
11 references
public double GetParameterValue(string parameterName)
{
    _parameter.Clear();
    parameterName = parameterName.Replace(oldValue: " ", newValue: string.Empty).ToLower();
    return dictDiffParam.ContainsKey(parameterName) ? dictDiffParam[parameterName].value : 0;
}

2 references
public double GetParameterMaxValue(string parameterName)
{
    parameterName = parameterName.Replace(oldValue: " ", newValue: string.Empty).ToLower();
    return dictDiffParam.ContainsKey(parameterName) ? dictDiffParam[parameterName].maxValue : 0;
}

0 references
public double GetParameterMinValue(string parameterName)
{
    parameterName = parameterName.Replace(oldValue: " ", newValue: string.Empty).ToLower();
    return dictDiffParam.ContainsKey(parameterName) ? dictDiffParam[parameterName].minValue : 0;
}
```

Figure 103: Dynamic Difficulty Adjustment Get Parameter Value, Max Value, Min Value

The figure shown above shows three methods that return the following: The Parameter's current value, the Parameter's maximum value, and the Parameter's minimum value.



```
2 references
public ParameterDataNeeded GetDataNeeded(string dataName)
{
    dataName = dataName.Replace(oldValue: " ", newValue: string.Empty).ToLower();
    return dictParamDataNeeded.ContainsKey(dataName) ? dictParamDataNeeded[dataName] : null;
}

//DataAdjusted tells the system that data has been adjusted
2 references
public void DataAdjusted(string dataNeeded)
{
    dataNeeded = dataNeeded.Replace(oldValue: " ", newValue: string.Empty).ToLower();

    if (!dictParamDataNeeded.ContainsKey(dataNeeded)) return;

    //check which _parameter has this needed data and adjust the _parameter's value
    var parametersCount :int = difficultyParameters.Count;

    for (var i = 0; i < parametersCount; i++)
    {
        if (!difficultyParameters[i].HasData(dataNeeded)) continue;
        var parameter = difficultyParameters[i];
        var count :int = parameter.dataNeeded.Count;

        for (var j = 0; j < count; j++)
        {
            var dataNeededName :string = parameter.dataNeeded[j].Replace(oldValue: " ", newValue: string.Empty).ToLower();

            if (dataNeededName != dataNeeded) continue;

            AdjustParameter(dataNeeded, parameter);
        }
    }
}
```

Figure 104: Dynamic Difficulty Adjustment: Get Data Needed, Data Adjusted

The figure shown above are methods that return the data needed, and a method

called DataAdjusted tells the system that there is adjusted data. If data is adjusted, it will check through which parameters need the data. If there are data during checking, it will adjust the parameter according to which the data needed was greater or less than its previous value.

```
1 reference
private void AdjustParameter(string dataNeeded, DifficultyParameter parameter)
{
    //check if we should increase difficulty
    if (dictParamDataNeeded[dataNeeded].IncreaseDifficulty())
    {
        try
        {
            _parameter.Add(parameter.name);
            parameter.AdjustDifficultyParameterValue(Difficulty.higher);
            ChangeDDAText();
        }
        catch
        {
            _parameter.Remove(parameter.name);
        }
    }

    //check if we should decrease difficulty
    if (!dictParamDataNeeded[dataNeeded].DecreaseDifficulty()) return;

    try
    {
        _parameter.Add(parameter.name);
        parameter.AdjustDifficultyParameterValue(Difficulty.lower);
        ChangeDDAText();
    }
    catch
    {
        _parameter.Remove(parameter.name);
    }
}
```

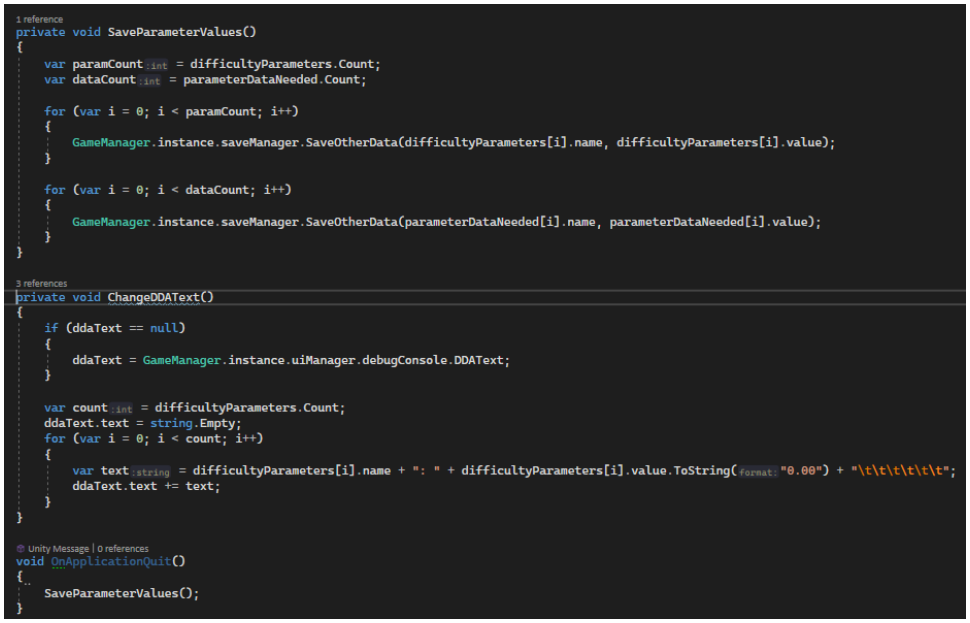
Figure 105: Dynamic Difficulty Adjustment: Adjust Parameter

The figure above shows a method to adjust the parameter. It will check if the parameter needs to increase its difficulty or decrease its difficulty. After checking, it will adjust if the methods from the ParameterDataNeeded class return true. The calling of the updates on the UI will then occur.

```
1 reference
public void ChangeValuesFromSaved()
{
    var paramCount :int = difficultyParameters.Count;
    var dataCount :int = parameterDataNeeded.Count;

    for (var i = 0; i < paramCount; i++)
    {
        if (!GameManager.instance.saveManager.LoadDataObject(difficultyParameters[i].name, out double value))
            continue;
        difficultyParameters[i].value = value;
    }

    for (var i = 0; i < dataCount; i++)
    {
        if (!GameManager.instance.saveManager.LoadDataObject(parameterDataNeeded[i].name, out float value))
            continue;
        parameterDataNeeded[i].value = value;
        parameterDataNeeded[i].previousValue = value;
    }
    ChangeDDAText();
}
```

Figure 106: Dynamic Difficulty Adjustment: Change Values From Saved

The figure shown above is for the persistence of the values of the parameters and the data needed. The calling of it occurs after the instantiation of the Player GameObject. It will go through all the data and load the value; if the save manager loads a value, the system will change it. For the data needed, the only difference is that it also changes the value of the previous value.



```
1 reference
private void SaveParameterValues()
{
    var paramCount:int = difficultyParameters.Count;
    var dataCount:int = parameterDataNeeded.Count;

    for (var i = 0; i < paramCount; i++)
    {
        GameManager.instance.saveManager.SaveOtherData(difficultyParameters[i].name, difficultyParameters[i].value);
    }

    for (var i = 0; i < dataCount; i++)
    {
        GameManager.instance.saveManager.SaveOtherData(parameterDataNeeded[i].name, parameterDataNeeded[i].value);
    }
}

3 references
private void ChangeDDAText()
{
    if (ddaText == null)
    {
        ddaText = GameManager.instance.uiManager.debugConsole.DDAText;
    }

    var count:int = difficultyParameters.Count;
    ddaText.text = string.Empty;
    for (var i = 0; i < count; i++)
    {
        var text:string = difficultyParameters[i].name + ": " + difficultyParameters[i].value.ToString(format: "0.00") + "\t\t\t\t\t\t\t";
        ddaText.text += text;
    }
}

Unity Message | 0 references
void OnApplicationQuit()
{
    SaveParameterValues();
}
```

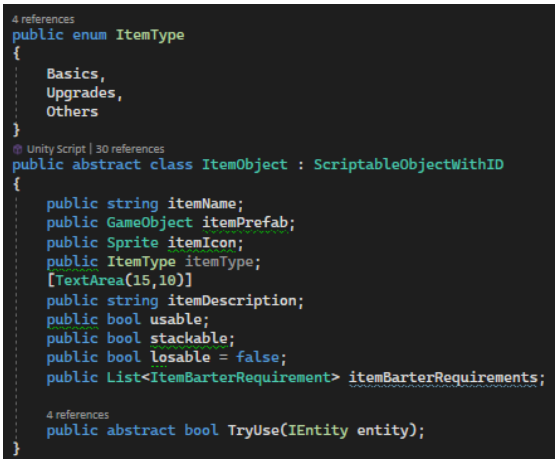
Figure 107: Dynamic Difficulty Adjustment: Save Parameter Values, Change DDA Text, On Application Quit

The SaveParameterValue method saves the value of the parameters and the data needed. The ChangeDDAText method is an update method for the UI. OnApplicationQuit runs when the application quits; if it does, then the SaveParameterValues method is run to save the values of the parameters and the data needed.

Items, Inventory, Bartering System

The Items, Inventory, and Bartering System in the project is a vital feature within the game. It is responsible for player rewards and player strategies during gameplay. A player will think about what items to keep or not to keep if they deem it necessary to

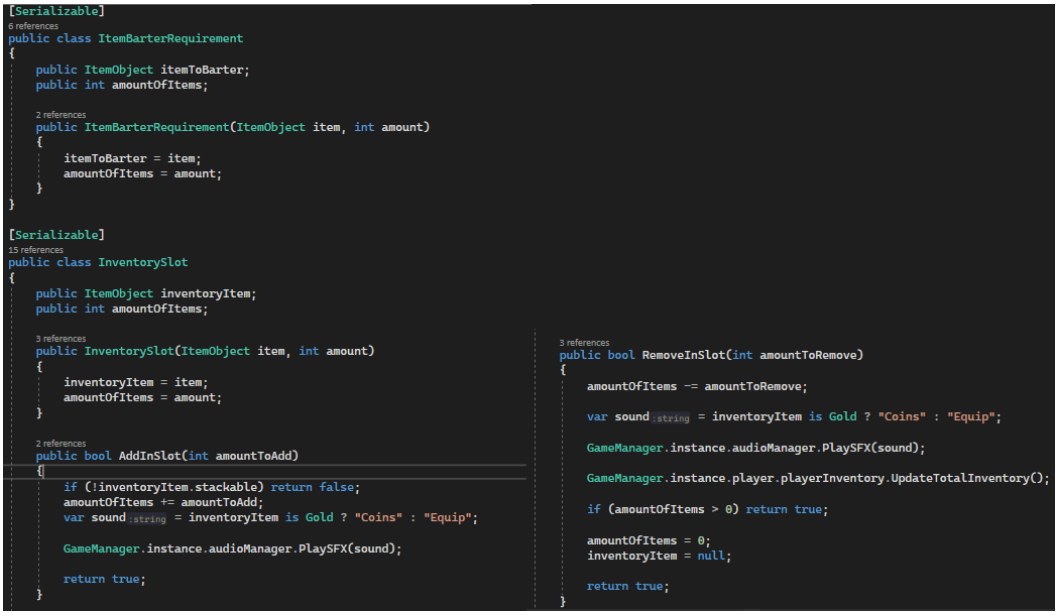
barter it with something more significant. The creators use the ScriptableObject feature in the game engine when creating the system.



```
4 references
public enum ItemType
{
    Basics,
    Upgrades,
    Others
}
Unity Script | 30 references
public abstract class ItemObject : ScriptableObjectWithID
{
    public string itemName;
    public GameObject itemPrefab;
    public Sprite itemIcon;
    public ItemType itemType;
    [TextArea(15,10)]
    public string itemDescription;
    public bool usable;
    public bool stackable;
    public bool losable = false;
    public List<ItemBarterRequirement> itemBarterRequirements;

    4 references
    public abstract bool TryUse(IEntity entity);
}
```

Figure 108: Items, Inventory, Bartering System: Item Object

The above figure shows the base class ItemObject. The container contains the item's name, the prefabricated object of the item, the icon of the item, the type of the item, the description of the item, booleans to check if it is a usable item or not, can be stacked, or not, or can lose after the death of the player. A list of ItemBarterRequirement is also in the ItemObject. Lastly, an abstract class returns a boolean if the usage of the object occurs.



```
[Serializable]
6 references
public class ItemBarterRequirement
{
    public ItemObject itemToBarter;
    public int amountOfItems;

    2 references
    public ItemBarterRequirement(ItemObject item, int amount)
    {
        itemToBarter = item;
        amountOfItems = amount;
    }
}

[Serializable]
15 references
public class InventorySlot
{
    public ItemObject inventoryItem;
    public int amountOfItems;

    3 references
    public InventorySlot(ItemObject item, int amount)
    {
        inventoryItem = item;
        amountOfItems = amount;
    }

    2 references
    public bool AddInSlot(int amountToAdd)
    {
        if (!inventoryItem.stackable) return false;
        amountOfItems += amountToAdd;
        var sound :string = inventoryItem is Gold ? "Coins" : "Equip";

        GameManager.instance.audioManager.PlaySFX(sound);

        return true;
    }

    3 references
    public bool RemoveInSlot(int amountToRemove)
    {
        amountOfItems -= amountToRemove;

        var sound :string = inventoryItem is Gold ? "Coins" : "Equip";

        GameManager.instance.audioManager.PlaySFX(sound);

        GameManager.instance.player.playerInventory.UpdateTotalInventory();

        if (amountOfItems > 0) return true;

        amountOfItems = 0;
        inventoryItem = null;

        return true;
    }
```

Figure 109: Items, Inventory, Bartering System: Item Barter Requirement, Inventory Slot

The ItemBarterRequirement class contains an ItemObject called ItemToBarter and the amount of which the requirements for the barter. It also has a constructor to store the reference of the newly instantiated ItemBarterRequirement.

The Inventory Slot is a slot that contains the item and the number of items in the slot. It has a constructor to store the reference whenever creating a new slot. Two methods inside the class are AddInSlot and RemoveInSlot, which adds the number of items in the slot and removes the number of items in the slot, respectively.

```
public class PlayerInventory : MonoBehaviour
{
    [SerializeField] private int _itemLimitPerSlot = 69;
    public List<InventorySlot> inventorySlots;

    private readonly Dictionary<ItemObject, int> _totalInventory = new Dictionary<ItemObject, int>();
```

Figure 110: Items, Inventory, Bartering System: Player Inventory

The Player Inventory class, which derives from MonoBehaviour, is a class attached to the player GameObject. The class contains how much would be the item limit per slot and a list of Inventory Slots. A read-only dictionary called _totalInventory is also present in the class for searching.

```
3 references
public void AddItemInPlayerInventory(ItemObject item, int amountToAdd)
{
    if (!item.stackable)
    {
        for (var i = 0; i < amountToAdd; i++)
        {
            var newInventorySlot = new InventorySlot(item, amount: 1);
            AddInEmptySlot(newInventorySlot);
        }
        return;
    }

    var slotCount = inventorySlots.Count;

    for (var i = 0; i < slotCount; i++)
    {
        var slot = inventorySlots[i];
        if (slot.inventoryItem != item) continue;

        if (item is Gold)
        {
            slot.AddInSlot(amountToAdd);
            amountToAdd = 0;
            break;
        }

        var targetAmount = slot.amountOfItems + amountToAdd;

        if (targetAmount <= _itemLimitPerSlot)
        {
            slot.AddInSlot(amountToAdd);
            amountToAdd = 0;
            break;
        }

        targetAmount -= _itemLimitPerSlot;
        slot.amountOfItems = _itemLimitPerSlot;
        amountToAdd = targetAmount;
    }

    if (amountToAdd > 0)
    {
        var newSlot = new InventorySlot(item, amountToAdd);
        AddInEmptySlot(newSlot);
    }

    UpdateTotalInventory();
    GameManager.instance.questManager.UpdateGatherQuest();
    GameManager.instance.uiManager.UpdateGoldUI();
    var player = GameManager.instance.player;
    GameManager.instance.uiManager.UpdateItemsUI(player.monsterManager, slot: -1, player.monsterSlots);
}
```

Figure 111: Items, Inventory, Bartering System: Player Inventory: Add Item In Player Inventory

The figure shown above displays the method called AddItemInPlayerInventory. The method adds an item to the player inventory, which automatically arranges the items in the inventory.

```
3 references
public void RemoveItemInInventory(ItemObject item, int amountToRemove)
{

    foreach (var slot in inventorySlots.Where(slot => slot.inventoryItem == item))
    {
        if (item is Gold)
        {
            slot.RemoveInSlot(amountToRemove);
            amountToRemove = 0;
            break;
        }

        var targetAmount :int = slot.amountOfItems - amountToRemove;
        if (targetAmount >= 0)
        {
            slot.RemoveInSlot(amountToRemove);
            amountToRemove = 0;
            break;
        }

        targetAmount *= -1;
        slot.amountOfItems = 0;
        amountToRemove = targetAmount;
    }

    if (amountToRemove <= 0)
    {
        UpdateTotalInventory();
        GameManager.instance.questManager.UpdateGatherQuest();
        GameManager.instance.uiManager.UpdateGoldUI();
        var player = GameManager.instance.player;
        GameManager.instance.uiManager.UpdateItemsUI(player.monsterManager, slot: -1, player.monsterSlots);
        return;
    }

    RemoveItemInInventory(item, amountToRemove);
}
```

Figure 112: Items, Inventory, Bartering System: Player Inventory: Remove Item In Inventory

The RemoveItemInInventory method shown above removes an item from the player inventory, automatically arranging the items in the inventory.

```
2 references
private void AddInEmptySlot(InventorySlot newSlot)
{
    var slotsCount :int = inventorySlots.Count;
    for (var i = 0; i < slotsCount; i++)
    {
        if (inventorySlots[i].inventoryItem != null) continue;

        inventorySlots[i].inventoryItem = newSlot.inventoryItem;
        inventorySlots[i].amountOfItems = newSlot.amountOfItems;
        GameManager.instance.questManager.UpdateGatherQuest();
        GameManager.instance.audioManager.PlaySFX(soundName: "Equip");
        break;
    }
    UpdateTotalInventory();
}

1 reference
public int GetTotalAmountItems(ItemObject item)
{
    return _totalInventory.TryGetValue(item, out var amount :int) ? amount : 0;
}

5 references
public void UpdateTotalInventory()
{
    var slotCount :int = inventorySlots.Count;
    _totalInventory.Clear();

    for (var i = 0; i < slotCount; i++)
    {
        if(inventorySlots[i].inventoryItem == null) continue;
        try
        {
            _totalInventory.Add(inventorySlots[i].inventoryItem, inventorySlots[i].amountOfItems);
        }
        catch
        {
            _totalInventory[inventorySlots[i].inventoryItem] += inventorySlots[i].amountOfItems;
        }
    }
}

2 references
public bool HasSufficientItem(ItemObject item, int amount)
{
    if (!_totalInventory.TryGetValue(item, out var currentAmount :int))
    {
        return false;
    }

    return currentAmount >= amount;
}
```

Figure 113: Items, Inventory, Bartering System: Player Inventory: Add In Empty Slot, Get Total Amount Items, Update Total Inventory, Has Sufficient Item

The figure above shows four methods namely AddInEmptySlot, GetTotalAmountItems, UpdateTotalInventory, HasSufficientItems. The following methods; adds a slot to an empty slot, return the total amount of items based on the item passed, update the total inventory Dictionary, and return a boolean if the player has sufficient items.

```
1 reference
public bool CanAdd(ItemObject item, int amount)
{
    if (item is Gold)
    {
        return true;
    }

    var canAdd = false;
    var slotCount :int = inventorySlots.Count;
    for (var i = 0; i < slotCount; i++)
    {
        var slot = inventorySlots[i];

        if (slot.inventoryItem == null)
        {
            canAdd = true;
            break;
        }

        if (!item.stackable) continue;

        if (slot.inventoryItem != item) continue;

        if (amount + slot.amountOfItems > _itemLimitPerSlot) continue;

        canAdd = true;
    }

    return canAdd;
}
```

Figure 114: Items, Inventory, Bartering System: Player Inventory: Can Add

The above method returns a boolean if the system can add items in the slot.

```
public class ItemDrop : MonoBehaviour, IInteractable
{
    [Layer][SerializeField] private int layer;
    [SerializeField] private float _timeToDisable = 5f;
    [SerializeField] private float _interactableDistance = 1f;
    [ReadOnly][SerializeField] private bool _isInteractable;

    private GameObject _currentPrefab;
    private Outline _outline;
    private ItemObject _item;
    private int _amount;
    private Player _player;
    private Transform _playerTransform;
    private Transform _thisTransform;
    private Vector3 _down = Vector3.down;
    private Vector3 _zero = Vector3.zero;
    private GameObject _thisObject;
    [ReadOnly] [SerializeField] private float _timeElapsed;
    private bool _prefabOn = true;
    private Coroutine _warningCoroutine = null;
    private ItemSpawner _spawner;
    private double _economyMax;

    private Dictionary<GameObject, Outline> _outlines = new Dictionary<GameObject, Outline>();
    private Dictionary<GameObject, Transform> _transforms = new Dictionary<GameObject, Transform>();
    private Dictionary<GameObject, Quaternion> _quaternions = new Dictionary<GameObject, Quaternion>();
```

Figure 115: Items, Inventory, Bartering System: Item Drop: Variables

The ItemDrop class derives from the MonoBehaviour class and IInteractable interface. These are items dropped in the overworld and to be picked up by the player. The class contains an integer of the layer, the time it will disable, the distance the object

can be interacted with, and a boolean to check if it is an interactable object. There are also variables needed for serialization to reduce garbage collection during runtime.

```
3 references
public void SetupItemDrop(Vector3 initPos, ItemObject item, int amount)
{
    if (item == null)
    {
        _thisObject.SetActive(false);
        return;
    }

    if (_thisObject == null)
    {
        _thisObject = gameObject;
    }

    if (_player == null)
    {
        _player = GameManager.instance.player;
        _thisTransform = transform;
        _playerTransform = _player.transform;
    }

    _item = item;
    _amount = amount;
    _spawner = null;
    _economyMax = GameManager.instance.difficultyManager.GetParameterMaxValue("Economy");
    Initialize(initPos);
}
```

Figure 116: Items, Inventory, Bartering System: Item Drop: Setup Item Drop

The calling of the SetupItemDrop method occurs whenever the instantiation of an ItemDrop occurs. It is to set up the item for player pickup.

```
1 reference
void Initialize(Vector3 initPos)
{

    var cast :bool = Physics.Raycast(origin: _thisTransform.position, direction: _down, out var hit, layer);
    if (!cast)
    {
        _thisObject.SetActive(false);
        return;
    }

    _thisTransform.position = initPos;
    StartCoroutine(routine: GoToPosition(hit.point, seconds: 1));
    _thisTransform.rotation = Quaternion.Euler(_zero);


    if (_currentPrefab != null && _transforms.TryGetValue(_currentPrefab, out var t :Transform))
    {
        t.parent = null;
        _currentPrefab.SetActive(false);
    }


    if (!_quaternions.TryGetValue(_item.itemPrefab, out var rotation :Quaternion))
    {
        rotation = _item.itemPrefab.transform.rotation;
        _quaternions.Add(_item.itemPrefab, rotation);
    }

    var obj :GameObject = GameManager.instance.pooler.SpawnFromPool(parent: null, _item.itemPrefab.name, _item.itemPrefab, _zero, rotation);

    if (!_outlines.TryGetValue(obj, out var outline))
    {
        outline = obj.GetComponent<Outline>();
        _outlines.Add(obj, outline);
    }

    if (!_transforms.TryGetValue(obj, out var trans :Transform))
    {
        trans = obj.transform;
        _transforms.Add(obj, trans);
    }

    trans.SetParent(_thisTransform);
    trans.localPosition = _zero;
    _currentPrefab = obj;
    _outline = outline;
    _timeElapsed = 0;
}
```

Figure 117: Items, Inventory, Bartering System: Item Drop: Initialize

The calling of the Initialize method occurs at the end of the SetupItemDrop method. The initialization of the item drop focuses on dropping the items at what position in the overworld by casting a ray downwards filtered with the layer. When there is no object cast, it will deactivate and ends the whole method. If it has, the object will be transferred to the initial position and then travels to the hit point in a second. The rotation is also set to zero. The method will then request the graphic in the pooler, and the item drop object will then be a parent to the graphic.

```
1 reference
public void SetupItemDrop(Vector3 initPos, ItemObject item, int amount, ItemSpawner spawner)
{
    SetupItemDrop(initPos, item, amount);
    _spawner = spawner;
}

Unity Message | 0 references
void Update()
{
    CheckInteraction();
    CheckTimeDisable();
}
```

Figure 118: Items, Inventory, Bartering System: Item Drop: Setup Item Drop, Update

Another SetupItemDrop method is present, and its only difference is to get the reference of the ItemSpawner. Another method shown in the figure is the Update method, which will check if the item can be interacted with by the player and check if the item will be disabled.

```
private void CheckTimeDisable()
{
    _currentPrefab.SetActive(_prefabOn);

    _timeElapsed += Time.deltaTime;

    var disableMultiplier :double = Math.Round(_economyMax - GameManager.instance.difficultyManager.GetParameterValue("Economy"),
        MidpointRounding.AwayFromZero);

    disableMultiplier = disableMultiplier < .5 ? .5 : disableMultiplier;

    var warning :double = _timeToDisable * disableMultiplier * .75f;
    if (_timeElapsed >= warning && _warningCoroutine == null)
    {
        _warningCoroutine = StartCoroutine(routine:Warning());
    }

    if (_timeElapsed < _timeToDisable * disableMultiplier) return;

    try
    {
        GameManager.instance.uiManager.itemDropUi.Unsubscribe(itemDrop:this);
        _thisObject.SetActive(false);
        _currentPrefab.SetActive(true);
        StopAllCoroutines();
        _warningCoroutine = null;
        _timeElapsed = 0;
    }
    catch
    {
        //ignored
    }
}
```

Figure 119: Items, Inventory, Bartering System: Item Drop: Check Time Disable

The figure shown above shows the CheckTimeDisable method, which activates the graphic if the graphic should be on, checks for the time elapsed after spawned, get the disable multiplier from the Economy parameter, multiply it to the warning, if the time is greater than the warning time then the Warning Coroutine starts. If the time elapsed is less than the time to disable multiplied by the disable multiplier, it returns. If it surpasses the said time, the cancellation of the subscription of the ItemDrop occurs in the ItemDropUI class in the UIManager, and the object deactivates.

```
private void CheckInteraction()
{
    var distanceToPlayer :float = Vector3.Distance( a: _playerTransform.position, b: _thisTransform.position);
    if (distanceToPlayer <= _interactableDistance && GameManager.instance.inputHandler.currentMonster < 0)
    {
        _isInteractable = true;

        GameManager.instance.uiManager.itemDropUi.Subscribe( itemDrop: this, _item, _amount);

        if (_outline != null)
        {
            _outline.enabled = true;
            _outline.OutlineMode = Outline.Mode.OutlineVisible;
        }
    }
    else
    {
        if (_outline != null)
        {
            _outline.enabled = false;
        }

        _isInteractable = false;
        GameManager.instance.uiManager.itemDropUi.Unsubscribe( itemDrop: this);
    }

    if (!GameManager.instance.inputHandler.interact || !_isInteractable) return;
    GameManager.instance.inputHandler.interact = false;
    _isInteractable = false;

    Interact(_player);
}
```

Figure 120: Items, Inventory, Bartering System: Item Drop: Check Interaction

Shown above is the CheckInteraction method, which checks if the object is within the player's distance. If it is, then it can interact. If the player presses the key to interact with objects, the calling of the Interact method from the IInteractable interface occurs.

```
4 references
public void Interact(Player player)
{
    if (!_player.playerInventory.CanAdd(_item, _amount))
    {
        GameManager.instance.uiManager.debugConsole.DisplayLogUI(_amount + " pcs. of " + _item.itemName + " will not be added. Insufficient inventory storage.");
        return;
    }

    if (_thisObject == null)
    {
        _thisObject = gameObject;
    }

    _currentPrefab.SetActive(true);
    StopAllCoroutines();
    StartCoroutine(routine:FollowPlayer(_playerTransform, seconds:1));
}
```

Figure 121: Items, Inventory, Bartering System: Item Drop: Interact

The addition of items occurs in the Interact method shown above. A UI log will be displayed if the system cannot add the item. If the system can add the item, then the item will follow the player.

```
1 reference
IEnumerator Warning()
{
    var elapsed = 0f;
    while (true)
    {
        elapsed += Time.deltaTime;
        if (elapsed < .75f) continue;

        _prefabOn = !_prefabOn;
        elapsed = 0f;
        yield return null;
    }
}

1 reference
IEnumerator GoToPosition(Vector3 toPos, float seconds)
{
    var timeElapsed = 0f;
    while (timeElapsed < seconds)
    {
        timeElapsed += Time.deltaTime;
        _thisTransform.position = Vector3.Lerp(a:_thisTransform.position, b:toPos, t:timeElapsed / seconds);
        yield return null;
    }
}

1 reference
IEnumerator FollowPlayer(Transform playerTransform, float seconds)
{
    var timeElapsed = 0f;
    while (timeElapsed < seconds)
    {
        _currentPrefab.SetActive(true);
        _thisTransform.position = Vector3.Lerp(a:_thisTransform.position, b:playerTransform.position, t:timeElapsed / seconds);
        GameManager.instance.uiManager.itemDropUi.Unsubscribe(itemDrop:this);

        if (Vector3.Distance(a:_thisTransform.position, b:playerTransform.position) <= .5f)
        {
            StopAllCoroutines();
            GameManager.instance.uiManager.itemDropUi.Unsubscribe(itemDrop:this);
            _player.playerInventory.AddItemInPlayerInventory(_item, _amount);
            GameManager.instance.audioManager.PlaySFX(soundName:"Confirmation");
            if (_spawner != null)
            {
                _spawner.RemoveItemInCurrentItems(_thisObject);
            }
            _thisObject.SetActive(false);
            _currentPrefab.SetActive(false);
            _transforms[_currentPrefab].parent = null;
        }

        timeElapsed += Time.deltaTime;
        yield return null;
    }

    _thisObject.SetActive(false);
}
```

Figure 122: Items, Inventory, Bartering System: Item Drop: Warning, Go To Position, Follow Player

The above figure shows three Coroutines called Warning, GoToPosition, FollowPlayer. The Warning coroutine loops through a specific time and turns the graphic on and off. GoToPosition Coroutine moves the object to a particular position. FollowPlayer Coroutine follows the player, and if the distance between the drop and player is less than .5 meters, then add the item to the player, unsubscribe from the ItemDropUI class, Play a sound effect, if there is a spawner, remove the item in the spawner, deactivate the object, the graphic, and set the parent to null.

```
public class ItemDropUI : MonoBehaviour
{
    public List<Button> buttons;

    private Dictionary<Button, GameObject> _objects = new Dictionary<Button, GameObject>();
    private Dictionary<Button, UITweener> _tweeners = new Dictionary<Button, UITweener>();
    private Dictionary<Button, TextMeshProUGUI> _texts = new Dictionary<Button, TextMeshProUGUI>();
    private Dictionary<Button, Image> _images = new Dictionary<Button, Image>();
    private Dictionary<Button, TooltipTrigger> _tooltipTriggers = new Dictionary<Button, TooltipTrigger>();
    private Dictionary<ItemDrop, Button> _itemDrops = new Dictionary<ItemDrop, Button>();
    private int _buttonsCount;

    Unity Message | 0 references
    void Awake()
    {
        if(_objects.Count > 0) return;

        _buttonsCount = buttons.Count;

        for (var i = 0; i < _buttonsCount; i++)
        {
            var obj :GameObject = buttons[i].gameObject;
            _objects.Add(buttons[i], obj);
            _tweeners.Add(buttons[i], buttons[i].GetComponent<UITweener>());
            _texts.Add(buttons[i], buttons[i].GetComponentInChildren<TextMeshProUGUI>());
            _images.Add(buttons[i], (Image)buttons[i].targetGraphic);
            _tooltipTriggers.Add(buttons[i], buttons[i].GetComponent<TooltipTrigger>());
            obj.SetActive(false);
        }
    }
```

Figure 123: Items, Inventory, Bartering System: Item Drop UI: Variables, Awake

The UIManager caches the ItemDropUI class. It contains a list of buttons and dictionaries where the button is the key. Serialization of the objects is the purpose of these dictionaries. The Awake method gets the component of the button, the UITweener, its text, and its tooltip trigger.

```
1 reference
public void Subscribe(ItemDrop itemDrop, ItemObject item, int amount)
{
    if(_itemDrops.ContainsKey(itemDrop)) return;

    for (var i = 0; i < _buttonsCount; i++)
    {
        var button = buttons[i];

        var tweener = _tweeners[button];
        var obj :GameObject = _objects[button];

        if (obj.activeInHierarchy) continue;


        _texts[button].text = "x" + amount;
        _images[button].sprite = item.itemIcon;
        _tooltipTriggers[button].SetTitleContent(title:item.itemName, content:item.itemDescription);
        button.onClick.RemoveAllListeners();
        button.onClick.AddListener(call:() => AddItem(tweener, itemDrop));
        _itemDrops.Add(itemDrop, button);

        obj.SetActive(true);
        return;
    }
}

4 references
public void Unsubscribe(ItemDrop itemDrop)
{
    if (!_itemDrops.TryGetValue(itemDrop, out var button)) return;

    GameManager.instance.uiManager.tooltip.HideToolTip();
    _itemDrops.Remove(itemDrop);
    _tweeners[button].Disable();
}

1 reference
private void AddItem(UITweener tweener, ItemDrop drop)
{
    tweener.Disable();
    drop.Interact(GameManager.instance.player);
}
```

Figure 124: Items, Inventory, Bartering System: Item Drop UI: Subscribe, Unsubscribe, Add Item

The Subscribe method checks which button is inactive. It will then activate the inactive button to display in the UI. The button will then be filled with data such as its amount, set the tooltip's title and content, add a listener to the button, add the button on click, and add the itemDrop, an ItemDrop class to the dictionary, then activates the button.

The Unsubscribe method removes itemDrop from the dictionary and disables the item UI.

The AddItem method is an OnClick function added to the item button, and when clicked, this method's invocation occurs. It will disable the tweener.

3D Modeling, Rigging, Animation

Producing the game requires excellent 3D modeling, rigging, and animation knowledge. This project's creators note that the project needs to be as efficient as possible. The greater the number of triangles in a 3D mesh, the more level of detail on a model, but it would be computationally intensive to display. The creators must reduce the number of triangles in a scene to decrease render times.

So, every time the creators model a new monster, they should consider the number of triangles in a model. The industry calls models with a relatively small number of polygons low-poly models. With this in mind, the creators limit the number of triangles: no model should amount to more than 5,000 triangles. The creators also used the program Blender for 3D modeling.

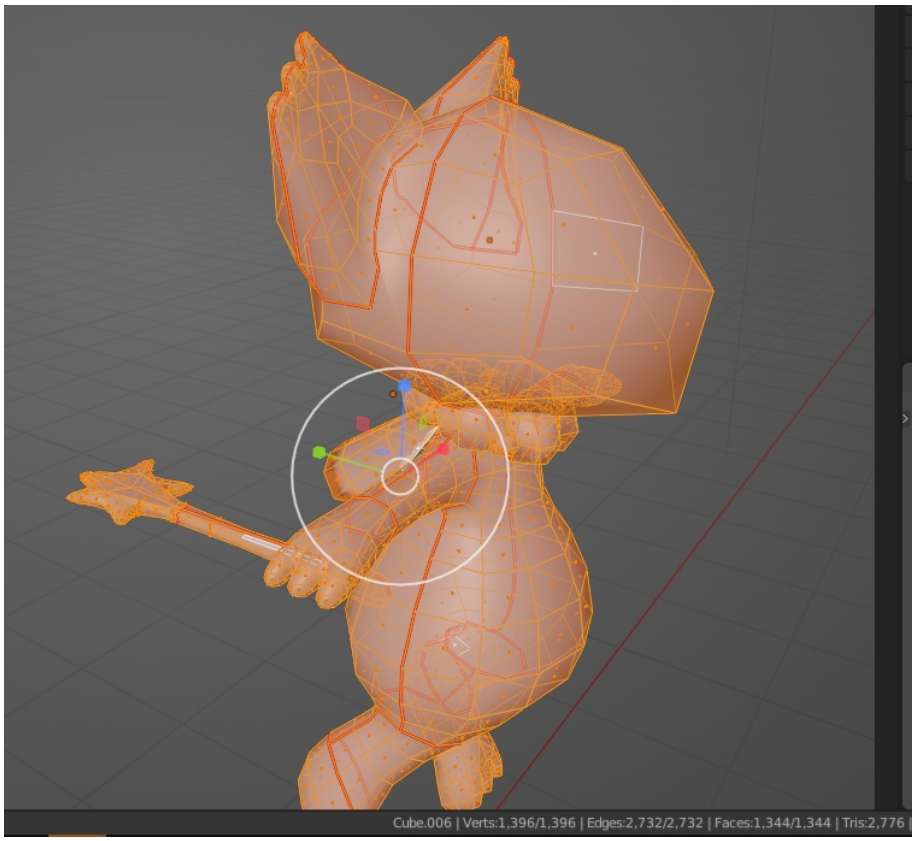


Figure 125: Chanaque: Its model only has 2,776 Triangles.

For rigging, the creators utilized the Rigify add-on in Blender. The Rigify add-on has pre-made bones for familiar creatures such as Humanoids, Cats, Dogs, and Horses. After aligning the bones to the model, Blender can make the bones a parent with automatic weights for the models to be animation-ready.

The creators have also decided to do rigging with Inverse Kinematics. Inverse kinematics is vital to game development and 3D movement, where it is utilized to interface game characters genuinely to the world. The project's creators can easily create animations by creating handles for the hand, feet, or tails, if any, and the rest of the body will automatically move by only moving the handles.

After creating Inverse Kinematics for the rig, the creators will animate the model. The creators should animate three actions for each monster: Idle animation, Move animation and Attack animation. During animating the model, the creators would like to note that each animation will have a separate Action in Blender. It is so the program can bake the animations on the model when exported. The exported model will have baked actions within the file, and when the creators import the model in Unity, the file will contain the said animations.

Humanoid creatures such as the Player characters, Unique NPCs, and Generic NPCs, are animated using Mixamo. Mixamo is a library of thousands of full-body character movements caught by proficient movement entertainers. An animation is repositioned to the creator's personality and can be reviewed and altered straightforwardly with Mixamo, so the creators have some control over the look and feel of each movement.

Post-Production

Game-Build Process and Exception Handling

During post-production, The creators built the game in WebGL. However, many hurdles happened during the building process. The creators did exception-handling to mitigate errors found. Later on, during exception-handling, the creators found out that the build size caused all the errors in the game. The creators discovered that the build size had exceeded 200MB, which is one of the requirements in building a stable WebGL build. The system did not instantiate objects intended to exist because the browser could not handle the number of objects needed in the game. Because of this, object references are missing from the components. The creators tried to compress the files, specifically the sound files, the models, and the textures, to their smallest file size. The creators changed the texture sizes to the power of two sizes to help the engine have an optimized compression for textures.

All these come with a cost: the quality. An extremely compressed build has made the textures unreadable and the model's colors unmatched because the 3D UVs not aligned to its actual texture. Accurate turned inaccurate 3D animations existed because of the optimized vertices. Sound has become distorted due to the compression. All these happened because of the compression process.

Because of all these, the creators realized that with the number of assets in the project: such as 18 monsters, with each monster having its texture, three animations per monster, with its 3D models, the nonexistence of a probability in loading all these in a browser have become a vision. Not to mention the 3D terrain model, item models with their texture and shaders, environment models, the NPCs textures, animations, model,

the player character's textures, animations models, UI textures, other textures, and various textures it has become impossible for a browser to render all these at once.

Because of this thought, the project's creators decided to build the project on a PC. With PC builds, it utilizes the PC's threads and memory, not relying on the browser's task-handling management. However, this significant decision does not stray away from the creators' purpose in building the project in a WebGL build in the first place. The project's purpose of being a WebGL build is to expand the demographic to a broader audience since they only utilize the browser in accessing the game. However, by providing the players a downloadable file to play with, they could download the game from the game hosting website where the creators published the game and play it using their computers.

Initial Usability Testing – Game Users Satisfaction Scale

Apart from the building process, the researchers conducted a survey called Game Users Satisfaction Scale (GUESS) wherein it is a tool that accurately quantifies video game satisfaction supported on fundamental components.

The survey begins by asking the player their identified gamer group (i.e., newbie/novice, casual, core/mid-core, and hardcore/expert). This way, the researchers will know the variety of the players in this study. The player's age also matters since, during the GUESS development, the study's researchers acknowledged gamers' ages in the study. The gamers' ages were at an average of 23.13, while the players in this study's survey were at 21.88, not as far as the former study's age group. The researchers chose this particular unit of players to guarantee that a broad scope of gamers with various gaming views and experiences can comprehend the form.

The researchers then asked the project's players to download the game via a game hosting website called itch.io. After download, the researchers ask them to play the game until they are satisfied with their exploration of the in-game world. After playing, the researchers again asked the players to answer the survey questions. The tool used for the survey is called Google Forms, which the researchers of this study used to create online forms and surveys with multiple question types. Since the GUESS developers recommend that survey respondents answer the questions randomly, the researchers set the forms to shuffled to randomize the survey questions. The online form contained demographic inquiries and orders of affirmation from the GUESS item pool on a 7-point, unipolar scale with response choices.

The GUESS improvement went through an intensive examination with over 1,300 members. Along these lines, the engineers of the said scale have distinguished nine factors that, taken together, give a total image of how charming a game is.

The following nine factors are: **Usability/Playability**, **Narratives**, **Play engrossment**, **Enjoyment**, **Creative freedom**, **Audio aesthetics**, **Visual aesthetics**, **Personal gratification**, and **Social connectivity**.

Usability or Playability is the straightforwardness of the game's players can learn how natural the game's point of interaction and menu framework is.

Narratives is the measurement of how charming the game's story components are and how well its developers created the characters concerning their in-game story.

Play engrossment is how well the game actuates a condition submerged, including forgetting about time while playing and fervor to play once more.

Enjoyment is how much the players have enjoyed the game.

Creative freedom is how well the game invigorates interest and considers creative control.

Audio aesthetics is how engaging players interpret the music and audio cues of the game.

Visual aesthetics is how engaging players interpret the overall visuals of the game.

Personal gratification is how propelling the game is to play and keep on playing.

Social connectivity is how well the game considers associations between genuine individuals playing.

Through the nine factors, the GUESS developers could develop multiple items for each factor. This study's researchers have extensively handpicked each question to find the game's ratings overall objectively.

Based on the GUESS developer's recommendation for practical use of the scale, future researchers should average the ratings of all the items per construct to obtain a mean score for each subscale. Moreover, future researchers should also calculate the average score of a subscale to acquire a total score of video game satisfaction. However, the researchers will also provide a Percent Agreement result based on the 7-point unipolar scale of each item and its average based on the nine factors.

Another factor to consider is that the researchers of this study took out both Social Connectivity and Audio Aesthetics of the nine-factor scale in this project. The game's nature of being a single-player game will affect the game's ratings on the Social Connectivity factor, while the game's music was from Kevin Macleod's music library, so

the researchers decided to take out the Audio Aesthetics factor, too. The project's creators are planning to create an original soundtrack for the game in the future.

Below are the survey results interpretation of the *initial testing*:

Table 5: Initial Usability Testing: % of Agrees per Item

Item	% of Agrees	Construct
I think it is easy to learn how to play the game.	94%	Usability/Playability
I find the controls of the game to be straightforward.	88%	Usability/Playability
I am in suspense about whether I will succeed in the game.	88%	Personal Gratification
I feel the game allows me to be imaginative.	94%	Creative Freedom
I think the game is fun.	100%	Enjoyment
I feel creative while playing the game.	100%	Creative Freedom
I always know how to achieve my goals/objectives in the game.	88%	Usability/Playability
I feel I can explore things in the game.	100%	Creative Freedom
I find my skills gradually improve through the course of overcoming the challenges in the game.	100%	Personal Gratification
I do not need to go through a lengthy tutorial or read a manual to play the game.	88%	Usability/Playability
I think the characters in the game are well developed.	94%	Narratives
I think the game is visually appealing.	100%	Visual Aesthetics
I feel successful when I overcome the obstacles in the game.	94%	Personal Gratification
I feel bored while playing the game. (Reverse)	71%	Enjoyment
I find the game's interface to be easy to navigate.	100%	Usability/Playability
I feel the game constantly motivates me to proceed further to the next stage or level.	100%	Personal Gratification
I enjoy the game's graphics.	100%	Visual Aesthetics
I enjoy the fantasy or story provided by the game.	100%	Narratives
I find the game's menus to be user friendly.	94%	Usability/Playability
I feel the game allows me to express myself.	100%	Creative Freedom
I am likely to recommend this game to others.	100%	Enjoyment
I feel the game provides me the necessary information to accomplish a goal within the game.	100%	Usability/Playability
I think the game is unique or original.	94%	Creative Freedom
I temporarily forget about my everyday worries while playing the game.	71%	Play Engrossment
I am very interested in seeing how the events in the game will progress.	100%	Narratives

Table 6: Initial Usability Testing: % of Agrees based on the 9 GUESS Factors

Construct	Average % of Agrees
Usability/Playability	93%
Narratives	98%
Play Engrossment	71%
Enjoyment	90%
Creative Freedom	98%
Personal Gratification	96%
Visual Aesthetics	100%
Overall	92%

For the initial testing, the researchers calculated the percent agreement by acquiring all item scores in the rating of 5 to 7 from the 7-point unipolar scale and obtaining the average from the total number of respondents.

Based on the interpretation of the results, it is easy to assume that with many players who like to collect items or collectibles, the Creative Freedom of this project would be as high as 98%. It was its aspect of giving the players the ability to play their collectibles in the form of monsters, or in this project's case, mythicas. By playing and being the mythicas, players have captured their imaginative boost of inspiration during playtime.

90% agreed that the players enjoyed the game as considering the charm of the mythicas and their distinct behavior gives them a variety of gameplay styles during their run on the game.

The researchers' extensive plan on the game's narrative has something to do with the result having 98% agreed that the game's narrative was well executed. Giving the players a mission right from the get-go was one of the factors that affected this

score. Having an ethical dilemma of choosing between harmony and liberty of the human-mythica relationship also affected this score.

96% agree that they want to play again if given another opportunity. Almost all of them agreed that the game personally gratified them. Because of its compelling narrative and its exciting and suspenseful gameplay, these were factors to consider that affected this score.

However, in the case of Play Engrossment, only 71% agreed that they were within the game. The researchers' interpretation suggests that this score was widely affected by the number of bugs found by players during gameplay and because of the game's lack of narrative content. Observing the players' reactions, they were surprised that the game had ended even before they submerged their minds into the in-game world.

During testing, Usability or Playability has come from a rocky start. During the first tests of the game, some players suggested that controls should be inclusive, especially newbie players who prefer the Arrow Key Controls over the usual WASD Control Scheme. After the players' suggestions, the project's creators added the Arrow Key Control Scheme. However, extensive instructions on the controls in the game during its first stages saved this from having a lower score, having a percent agreement score of 93%.

Another factor to consider that the game's Usability or Playability score was this high is because of its AI for Dynamic Difficulty Adjustment feature. The three parameters that affect the game's difficulty, namely the game's pace, the monster's strengths, and the world's economy, affected their decisions on the game's playability,

not to mention the wide range of the player's gaming experience.

Lastly, all respondents agreed that the game's visuals were highly respectable. The researchers have chosen a color palette that complements each other. By choosing the right colors within the project, considering the world's lighting, creating particle effects that complement its art style, and detailing the animations of the world objects, the project's visual aesthetics have come a long way.

Overall, the percent agreement score of the game during initial testing is 94% which almost all agreed that this game would fulfill the player's needs of gratification and joy.

Final Usability Testing – Game Users Satisfaction Scale

Because of the player’s feedback and bugs seen, the creators of this project fixed the extreme issues that could potentially affect the player’s game experience. Hence, after the initial testing, the game’s creators have decided to do a final run of the usability testing. The final testing’s process is similar to the initial testing’s process.

Before the researchers present the results, they must present the descriptions of the final usability testing demographic. Explained below are the final usability testing demographic’s descriptions.

Table 7: Final Usability Testing: Player Demographics

Player Demographics	
Type	**Age**
Casual	21
Casual	21
Casual	21
Newbie or Novice	23
Newbie or Novice	20
Midcore or Core	21
Newbie or Novice	22
Hardcore or Expert	22
Casual	19
Midcore or Core	23
Hardcore or Expert	19
Midcore or Core	20
Casual	20
Midcore or Core	21
Casual	22
Casual	19
Newbie or Novice	22
Casual	20
Casual	23
Hardcore or Expert	20
Casual	20
Midcore or Core	22
Age Average:	20.95

It is important to note that the demographic's average age is 20.95, or in the age range of 19 to 23. The age ranges are essential because they fit the game's target demographic.

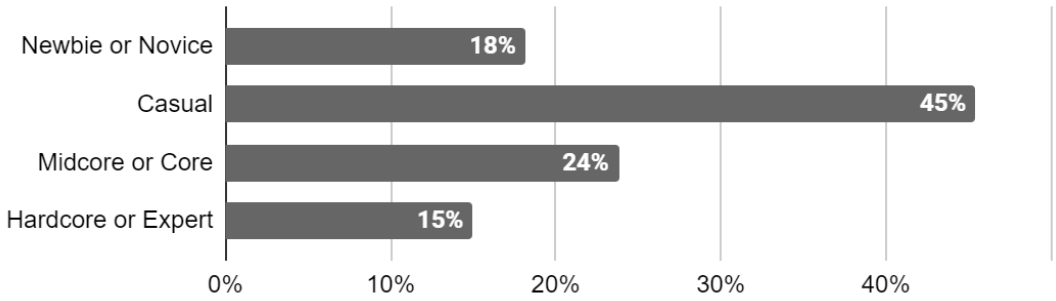


Figure 126: Final Usability Testing Demographic: Gamer Type

However, the researchers note that only 18% of them are newbie or novice gamers which would stray away from the project's purpose of testing its main feature, the AI for DDA. Admittedly, it is interesting to check the said feature's result during testing when 45% of the population of gamers are casual, and 24% are mid-core or core gamer participants. Undoubtedly, the researchers' prediction of the results is that the AI for DDA would feel more likely by the demographic of the final usability testing.

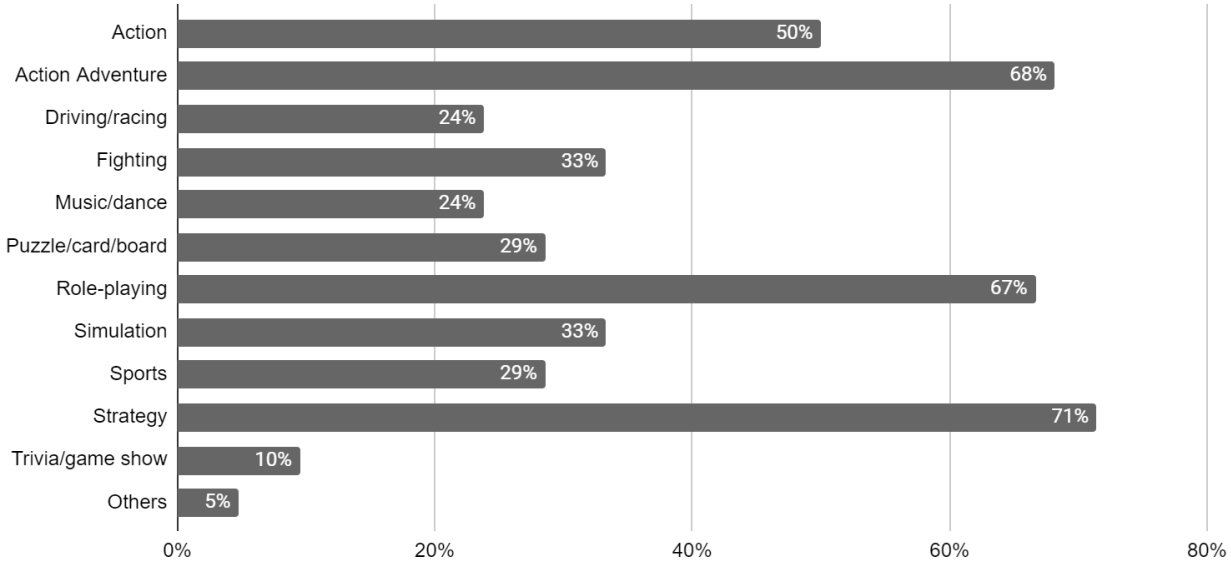


Figure 127: Final Usability Testing Demographic: Genre Representation

Following that thought, most of the final usability testing participants prefer to play Strategy games, precisely 71% of them. Due to this idea, the players might have

difficulty understanding the concept of an Action-RPG combined with the Monster-taming subgenre. However, it is essential to note that 67% of the respondents usually play Role-playing games, giving them at least an idea of how these games do. To explain this further, Western RPGs are different from JRPGs, with which the game in this study is associated. There might be nuances from the western RPG genre that could directly affect the survey results. However, the game genres presented above came from the GUESS developer's study. The genres Different famous gaming sites were counseled during the game genre organization process.

To note all these, the researchers present the results and interpretations of the final usability testing:

Table 8: Final Usability Testing: % of Agrees per Item

Item	% of Agrees	Construct
I think it is easy to learn how to play the game.	68%	Usability/Playability
I find the controls of the game to be straightforward.	73%	Usability/Playability
I am in suspense about whether I will succeed in the game.	50%	Personal Gratification
I feel the game allows me to be imaginative.	64%	Creative Freedom
I think the game is fun.	77%	Enjoyment
I feel creative while playing the game.	59%	Creative Freedom
I always know how to achieve my goals/objectives in the game.	50%	Usability/Playability
I feel I can explore things in the game.	77%	Creative Freedom
I find my skills gradually improve through the course of overcoming the challenges in the game.	59%	Personal Gratification
I do not need to go through a lengthy tutorial or read a manual to play the game.	45%	Usability/Playability
I think the characters in the game are well developed.	50%	Narratives
I think the game is visually appealing.	73%	Visual Aesthetics
I feel successful when I overcome the obstacles in the game.	68%	Personal Gratification
I feel bored while playing the game. (Reversed)	50%	Enjoyment
I find the game's interface to be easy to navigate.	64%	Usability/Playability
I feel the game constantly motivates me to proceed further to the next stage or level.	68%	Personal Gratification
I enjoy the game's graphics.	82%	Visual Aesthetics
I enjoy the fantasy or story provided by the game.	73%	Narratives
I find the game's menus to be user friendly.	50%	Usability/Playability
I feel the game allows me to express myself.	45%	Creative Freedom
I am likely to recommend this game to others.	68%	Enjoyment
I feel the game provides me the necessary information to accomplish a goal within the game.	55%	Usability/Playability
I think the game is unique or original.	64%	Creative Freedom
I temporarily forget about my everyday worries while playing the game.	32%	Play Engrossment
I am very interested in seeing how the events in the game will progress.	73%	Narratives

Table 9: Final Usability Testing: % of Agrees based on the 9 GUESS Factors

Construct	Average % of Agrees
Usability/Playability	83%
Narratives	65%
Play Engrossment	73%
Enjoyment	65%
Creative Freedom	83%
Personal Gratification	61%
Visual Aesthetics	93%
Overall	75%

In the final testing, the researchers calculated the percent agreement by acquiring all item scores in the rating of 5 to 7 from the 7-point unipolar scale and obtaining the average from the total number of respondents, similar to the initial testing.

Based on the interpretation of the results, it is essential to note that the demographics of this study have affected the results. It is easy to assume that with many players who like to collect items or collectibles, the Creative Freedom of this project would be 83%, compared to the initial testing where players of the said testing, as interviewed, like playing games with the genre the same as this study's game genre. Even considering the testing demographics, the project's Creative Freedom aspect still went far above the target of 75% of the researchers.

However, with players preferably playing games in the strategy genre, only 65% agreed that they enjoyed it. The game's Enjoyment aspect is heavily relative to the demographic's preferred games.

In the initial testing, the researchers surveyed in a face-to-face setup in which the project's creators explained the narrative in great detail. However, the online setup of

the final usability testing has not enabled the researchers to explain the narrative to the participants extensively. Considering the project is still a work-in-progress, the project's Narrative aspect still resulted in 65%.

Only 61% agree that they want to play again if given another opportunity. With the players preferably playing strategy games, it still garnered the said amount because of its compelling narrative and its exciting and suspenseful gameplay; these were factors to consider that affected this score.

However, it came as a surprise that 73% of the players agreed that they were within the game, compared to the initial testing's 71%, which the former is slightly higher. The researchers' interpretation suggests that this score increased since the creators have fixed minor issues in the project after initial testing.

Similar to the initial testing, Usability or Playability has come from a rocky start in the final testing. Players suggesting creating more settings in the controls have greatly affected the player's decisions in the survey. However, the researchers feel that the Usability or Playability aspect still resulted in 83% even with these factors. The results were due to its inclusive feature of AI for DDA.

Lastly, most respondents agreed that the game's visuals were highly respectable, precisely 93% of them. Same with the initial testing, the reason for this is that the researchers have chosen a color palette that complements each other. By choosing the right colors within the project, considering the world's lighting, creating particle effects that complement its art style, and detailing the animations of the world objects, the project's visual aesthetics have come a long way.

Still considering the demographics of the final testing, the percent agreement

score of the game is 75% which still barely passed the satisfactory average for the GUESS.

AI for DDA Testing

Hidden within the GUESS survey is an item which specifically measures if the players could notice the game's Dynamic Difficulty Adjustment. The item is engineered to disguise within the GUESS survey so players would not think that the feature's capability is also measured. The item is added after initial testing and the item's data were populated during the final testing. Below is the statement of the item:

I notice that the game's difficulty dynamically adjusts during gameplay.

Table 10: AI for DDA Testing: % of Agrees

AI for DDA Testing	**Data**																							
I notice that the game's difficulty dynamically adjusts during gameplay.	7	6	5	5	5	6	7	4	5	3	3	5	4	4	5	5	5	5	6	5	5	7		
% of Agrees	**77.27%**																							

The percentage of Agrees in the item states that 77.27% agreed that the project's AI for DDA is noticeable. The result passing the satisfactory average of percentage signifies that the game's AI for DDA is functional, and the participants feel that the game's difficulty shifted during gameplay. The practical use of the feature has benefited the participants, and it has come to good use.

CHAPTER IV

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

Summary of Findings

Based on the results, the factors that made this game respectable are its Visual Aesthetics, Creative Freedom, Narrative, Personal Gratification, and Usability or Playability.

The game's graphics were not that much, but choosing the right art style for the right demographics will make the game pop. Soft colors and pleasing to the eyes were two of many comments during the player's gameplay.

Its concept has made the player's imaginative freedom shine, making the creative freedom of this project a staple.

Planning the narratives from start to finish and having a vision of the end also made the narrative unique, as much of the games' narratives associated with the genre were straightforward. However, in this project's case, an ethical dilemma was established: either choose harmony between humans and mythicas or liberate the creatures from the horrors they faced because of humans.

The project's player's gratification has been affected by its ability to capture and be the monsters they have tamed. Transforming into a powerful beast from a helpless human has made the players' personally gratified.

The last factor that made this game great is its Usability and Playability—having parameters that affect the game's world overall has made the game easily accessible to players of different experiences.

However, from a production standpoint, creating a monster-taming action RPG is a huge undertaking. It takes excessive planning from its pre-production stage, from its world-building narrative to preparing the different systems needed in the project.

3D real-time monster-taming action RPGs take numerous assets. These assets include textures, sound, and 3D models. Because of this, memory usage and file size should also be kept in mind during production, especially in a WebGL project.

The utilization of Unity's ScriptableObject feature was the primary basis for systems as its usage provides less memory usage. Considering that the researchers need to establish a system of pooling objects called Object-pooling as instantiating objects beforehand and reusing them is an effective way to decrease the usage of memory in the device. The creators should also model low-poly 3D models in modeling the different monsters and characters in the project. The creators should also compress sound files to their lowest amount, and textures should be in sizes of the power of two since Unity's import manager effectively compresses textures in the said sizes.

Conclusions

Based on the results, the AI for DDA within the project has helped with the game's playability. Given the wide range of players with different gaming experiences, they could play the game without complaining of the game being too hard or too easy. With the researchers considering their reactions to the game, the controls have also been a significant factor that affects a game's difficulty.

Another thing to consider is that having a distinct AI behavior and the monsters' different stats to factor out could also affect the player's enjoyment of the game. The game has brought the players' imaginative inspirations to life because of its different

monsters, which affect the game’s world-building. So, the researchers conclude that future game developers should consider having distinct AI behavior, in this study’s case, bespoke behavior, combined with attribute-based playable characters.

Moreover, it is interesting to note that while some players may be interested in turn-based games, the creators should not ignore the player's reactions towards playing the project, which is leaning more toward being action-driven. Playing a real-time action game rather than a turn-based strategy game, which is a mechanic heavily associated with the genre, would give the players suspense during gameplay and motivate them to think about what actions they would do immediately to escape the game's end. The creators noted that action RPGs are better than turn-based games, which is one of the inadequacies that is pretty much associated with this project's primary genre.

The creators should directly market the game to its target demographic based on the final testing results. Players who are inclined to play Monster-taming Action-RPGs should be a factor when presenting the game to an audience. A game in this niche will confuse gamers not inclined in the genre.

Recommendations

With that, even with the project following all the processes and precautions to achieve a specific file size, the project still exceeded the recommended file size of a WebGL build.

Websites that host video games such as itch.io recommend that WebGL builds should not contain more than a thousand files after extraction of the compressed build, the overall size of the build should not be greater than 500MB, and a single extracted file should not be greater than 200MB.

While the final build may not have contained more than a thousand files and the overall size may not be greater than 500MB, Unity's WebGL build process collects all data within the project and store it into one file after build, which means one file from the build contains more than 200MB which this file contains almost all the data in the project.

How does this all add up? WebGL builds are not recommendable when it comes to creating monster-taming action RPGs. With the number of compressed assets to fit the recommended requirements for WebGL builds, the quality of the game will be affected. Small projects are fit with WebGL, but massive projects will be for PC.

However, even if the project is for PC, file compression was still helpful in mitigating performance issues, especially when rendering textures, as there are lesser draw calls due to its complete compression.

With all these in mind, even with the different features added to the project, players would recommend having more hints about what they should do within the game. However, having an open world would give them plenty of opportunities to explore the world and experience the stories behind each character. The narrative also hints at how they should decide what to do within the game. It is recommendable to give an option whether to have hints or indicators so players who prefer these kinds of gameplay will have the option to have them.

Another thing to recommend is that controls should be inclusive. Standard controls such as [WASD] may be preferable to many players, but some would want to have the arrow keys instead. Because of this, players will have inclusive control keys to what they prefer.

The researchers also recommend ScriptableObjects to make the project expandable. The researchers also suggest having a multi-scene setup for objects to have complete independence from each other. This way, the creation of different places within the game will have a certain kind of independence.

BIBLIOGRAPHY

[1] Marcel Tau, “Arcana Game Design Document,” Marcel Tau, February 2012. [Online]. Available: http://marceltau.com/files/Arcana_Project_GDD.pdf. [Accessed: July 2021]

[2] Unity Technologies, “ScriptableObjects,” Unity Documentation, 2 June 2021. [Online]. Available: https://docs.unity3d.com/es/current/Manual/class-ScriptableObject.html. [Accessed: July 2021]

[3] Bulbagarden, “Experience,” Bulbapedia, 16 February 2005. [Online]. Available: https://bulbapedia.bulbagarden.net/wiki/Experience. [Accessed: August 2021]

[4] Fandom, Inc., “Attributes,” Genshin Impact Fandom, 31 March 2020. [Online]. Available: https://genshin-impact.fandom.com/wiki/Attributes. [Accessed: August 2021]

[5] Mohammad Zohaib, “Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review,” Hindawi, 1 November 2018. [Online]. Available: https://downloads.hindawi.com/journals/ahci/2018/5681652.pdf. [Accessed: August 2021]

[6] Phan, Mikki & Keebler, Joseph & Chaparro, Barbara, “The Development and Validation of the Game User Experience Satisfaction Scale (GUESS),” *Human Factors: The Journal of the Human Factors and Ergonomics Society,* September 2016. 58. 10.1177/0018720816669646.

[7] Barbara Chaparro & Joseph Keebler, “Taking the GUESSwork out of video game satisfaction,” The Conversation, 16 September 2016. [Online]. Available: https://theconversation.com/taking-the-guesswork-out-of-video-game-satisfaction-65288. [Accessed: May 2022]

APPENDICES

Player Feedback

Initial Testing Feedback

Have you encountered any bugs or errors?	**Any comments and suggestions? You may write it down below.**
Just visual bugs	add moree mythicas hehe
Yes. Ask james.	
I can't get into the next stage saying "you can't interact why a mythica sees you" after I go back to the main menu and continue. The mythicas are also just ignoring the player.	the walls on the southern sides of the camera should be remove to give player a visibility on the environment
The interaction indicator "E" appeared for a while but there were no items around me.	The game is very fresh and has great potential. I hope you develop this further and release it to the market.
none	Character is hidden with the (rock) wall while walking on a horizontal path.
randomly falling off in the map	
I was forcing the character to run through the wall and it went through and falls, but there is an unstuck button to solve the problem so I'm not sure if it is really a bug. Another one is a visual bug during the part where you talk to Sulayman.	Add settings in the main menu to make the players adjust his/her preferred settings.
NA	
N/A	More maps and Mythicas to be available as the game expands in the future!
-the walls are covering the character; it's hard to play whenever I'm in battle with other mythicas because I can only see outlines. -there are minor glitches (falling character, letter E appears even when there's nothing to claim/pick up)	Great animation and graphics. The story is very entertaining and creative. There's a lot of potential in this game. Congratulations!!
NA	
The lowermost dungeon, left area (upon starting the game from first pillar), when you try and move through the walls there's a bug where you keep on falling even after restarting the game (only option is to "unstuck.").	(1) The game may consider adding a lock-on feature on mythicas (or any monster). (2) Character movement too fast-paced, might consider adding an option for adjusting movement. *sound design is great.
NA	:))))))))))))))))))
NA	NA
NA	Atleast one special skill when it transform into a other creature.

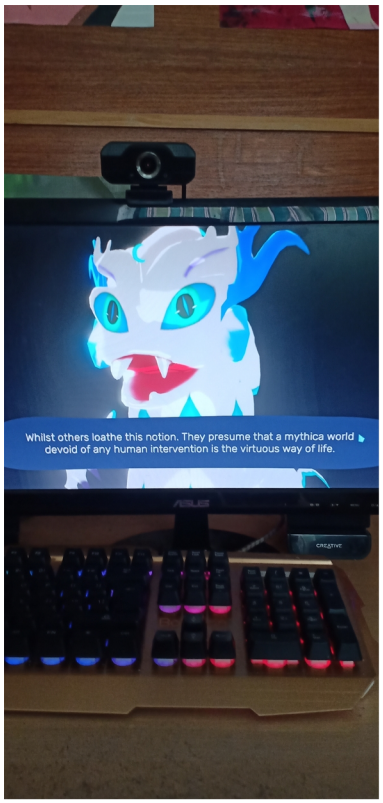
Final Testing Feedback

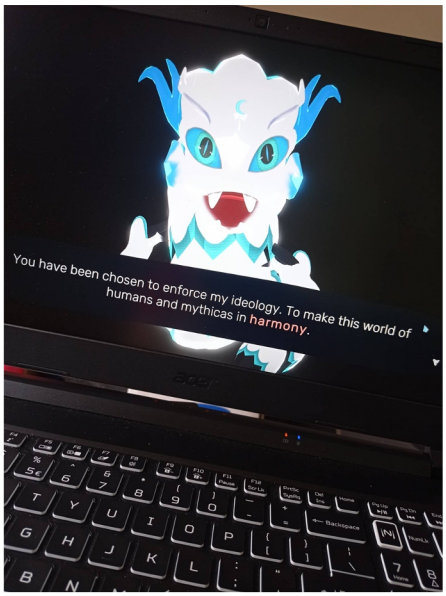
(1)After capturing my mythica, there was a movement glitch while the player guide was speaking. (2)After selecting the "Type Advantage" option, I couldn't press anything and needed to restart the game.	I enjoyed the game from it's story, combat, and visual appeal. However, there are many improvements that can be made which I believe would really have a positive impact in the game. (1)The player controls in which you need to press the ` key to switch from your mythica to your original character. This makes it harder for players like me that has a 60% keyboard needing to press the Fn key + ` key at the same time. It makes it harder to switch between mythica and original character during combat as well. (2)The mythica switching during battle or combat is a huge plus for me, however, disengaging doesn't seem to be an option as the enemies track you from far away. Maybe make it so that AI only tracks you in a certain distance? (3) I prefer that quest have a guide pathway or arrow that points to the area you need to go or npc you need to talk to. (4)When moving the player up and down, the camera seems to lag behind a bit. Considering this is a top down view, moving up and down creates clumped vision and players cannot see the direction they are going when moving up and down. (5) The mythica list could have pictures of the mythicas included. Overall, I want to see and play more from the game. Good luck!
at first playthrough I cant go through the white stuff that allows me to wake up then I restarted the game and I got out of the first area	There should be a settings button at the main menu
- When I was doing the first quest, the mythica is not responding and it just sits still while I was attacking or taming it. - When taming a mythica I sometimes get some random mythica instead of the one I was fighting ex. I was taming a chuchu but when the pop up came to announce what i captured it says antslayer. - When forcing to walk thru a cliff you can easily clip through the world	Add more mythicas, targeting system, slower walking speed and quest indicator.
NA	none
NA	NA
The place where it introduces classes, when I get there first before going to the left side and I activate a dialog sequence. The game will freeze at the end of the dialog.	I love it, the visuals are very poppy and smooth.
N/A	N/A
none	none
NA	Its good man
I cant change mythicas because i'm locked through the first 4 mythicas I unlock. Mythica Slot feature doesn't work.	Add combat skills

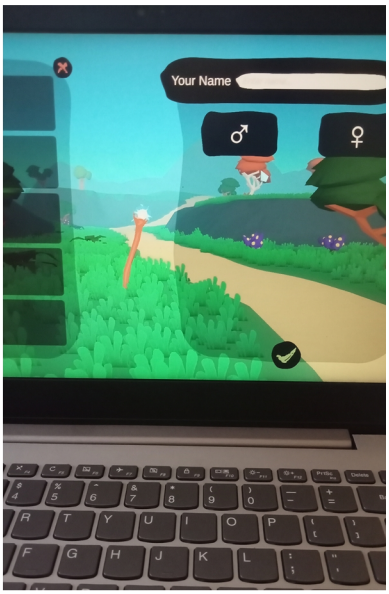
NA	None
NA	NONE
NA	
The character got stuck	Please fix the bugs
When I continue playing all I see is the UI and everything is black	I suggest add more instructions or tutorials.
NA	
yes, it seems that from the picture i have taken the game just suddenly could no longer register my clicks to continue the story line and yes i tried it many times and waited for 5mins. and it still didn't budge and if you think that the game is frozen or just hang/stop working no it didn't the background fire continues it's animation i tried looking with task manager and the game is running fine and when i restarted the game it just made me teleport to a not so far location of my last location and is force to fight a mythica.	The game is quite fun and as i can see it follows the same strategy as pokemon games, i assume that it is a inspiration game from pokemon. i haven't reach the outmost part of the game but i don't just like the idea of the dub of the characters, too static and incomprehensible maybe it's good when you do it on the first time encounter of the character of the mythica god but as of the same voice over over again from the begining it doesn't just make sense since mythica god should have a different language and so does the main character, sure it would best to do the incomprehensible voice but please don't repeat the voice from line to line since it's too static and it hurts the ears. the introduction of the game was quite scary since the mythica god just suddenly pop out slowly from the darkness which is not a good idea since i can see that the game focused on attracting kids and i suggest that the character should not slowly appear in darkness but lightness slowly comes out and takes over the darkness with the character itself appearing above to center since it's a god descending below the ground.
NA	Maybe add more distinction with regards to the design of the male and female avatar. It would be nice to have a slightly bigger map in the hud as well.
N/A	The sound or audio from the characters need to be improved, I think it will be better if it will be voiced by a person.
When I opened the game, the contents are not showing	
Yes. Constantly falling under the map. - It can be caused by dashing to the walls. Still In combat even though that there are no monsters nearby - -It happened by hitting the untamed monsters near the starting area. I had to quit and reopen the game and can't interact with the portal before waking up. Monster aggro is not gone even though you are very far away Border at the end of the map has no invisible walls.	Any comment and suggestions: Sounds: The sounds are not bad, and not good either. It is decent. Gameplay: The gameplay is kind unique to me to play or watch which is pretty much original on my end. It is similar to pokemon legends arceus, but with ability to change forms. The most fun thing I have in the game is actually not the battle gameplay itself, but with gameplay that you go to the top of the mountain and constantly used the dash button to feel like you are gliding in the air. The monsters range detection is very bad in that they would notice you from pretty far away even though you don't see them in the screen. I would like to have a map button to navigate the map easier . Mythika Combat and Gameplay: The combat feels

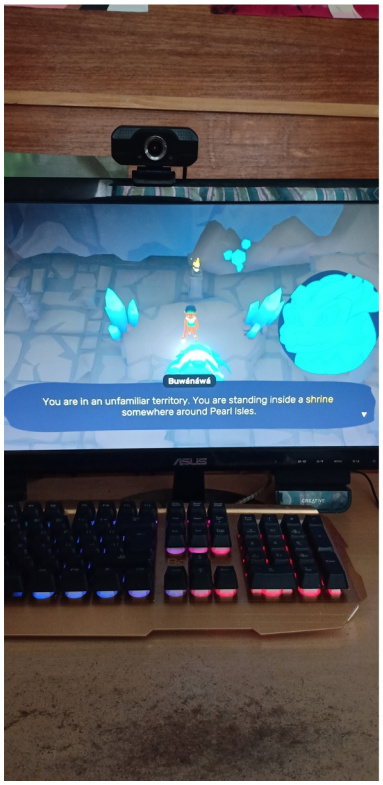
	smooth and doesn't lag. The different types of mythicas seems that they offered different kinds of gameplay, however based on 5 mythicas I have, somewhat they feel and look the same. I can't tell whether the bat mythika is a stinger when it feels like a bruiser. Somewhat lacking distinction between them. And I dislike that the fact it seems that I can't access the extra mythikas in the storage or the mythikas somewhat feels eternal in that I don't feel that I lose them permanently. Story: I can't say about the story, however the concept is somewhat interesting. One thing I like to see change is the inconsistency of words. Like some items are english, but some items are tagalog like 'mais'. It would be nice to be consistent like Apple = Mansanas, Coconut = Buko or Lubi. Character Designs: The character is decent. At least it is unique on that world and setting. Mythica Designs: The designs are decent. Unstuck Button: This button is really helpful especially I'm constantly stuck in bugs and errors.

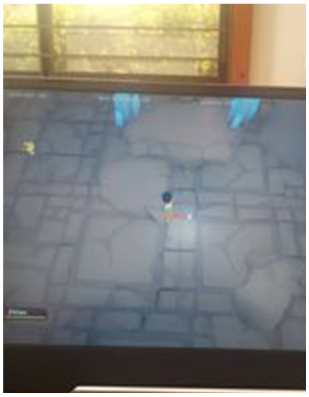
Documentation

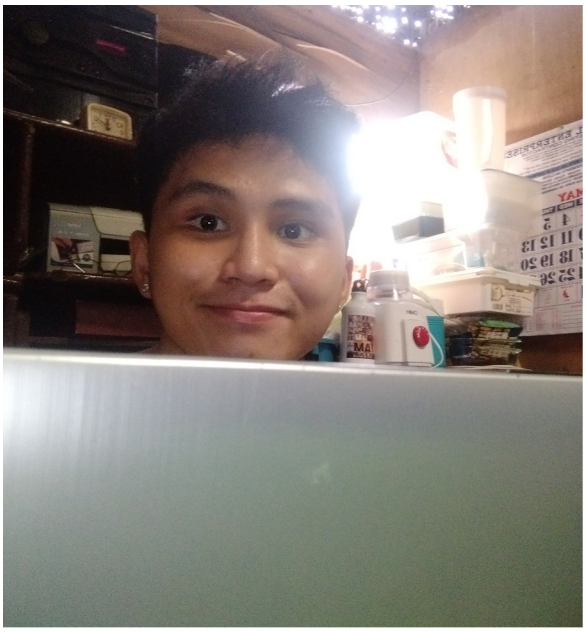


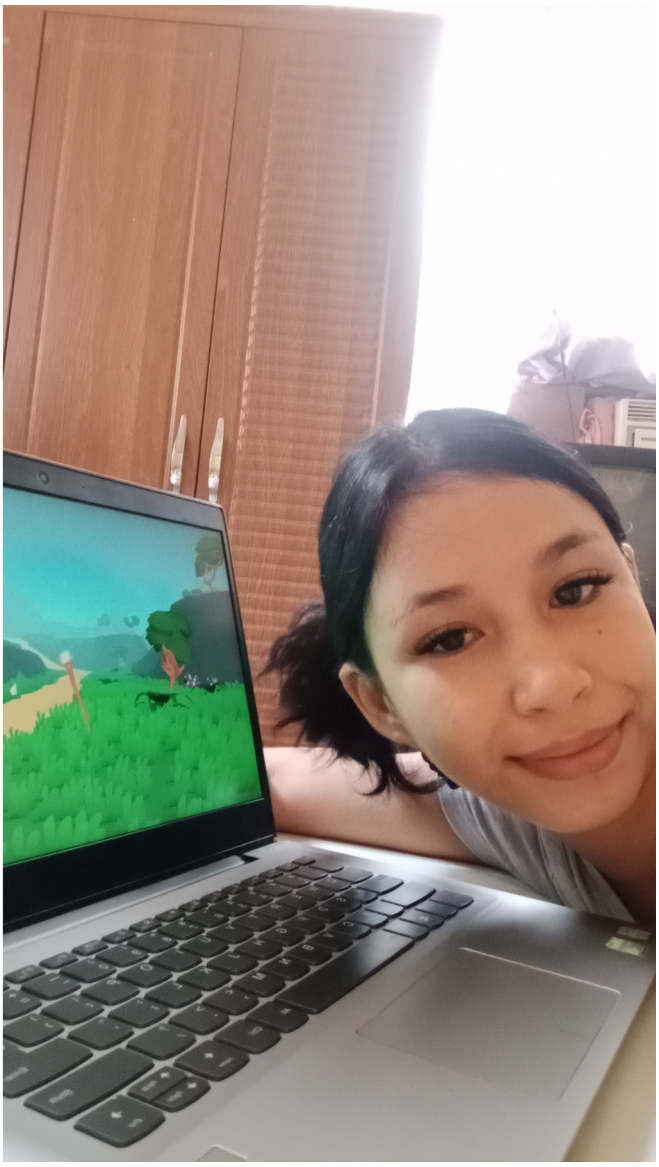












Curriculum Vitae

PERSONAL INFORMATION

Name	: **Dave Mossess L. Poro**	
Nickname	: **Dave**	
Date of Birth	: **April 24, 2000**	
Address	: **79-A Lorega St., Cebu City**	
Status	: **Single**	
Citizenship	: **Filipino**	
Religion	: **Roman Catholic**	
Email Address	: **deybpors@gmail.com**	
Contact Number	: **(+63) 9610285028**	

ACADEMIC INFORMATION

Tertiary	: **University of San Jose-Recoletos**	**(2018-2022)**
Secondary	: **Asian College of Technology – IEF**	**(2016-2018)**
	Abellana National School	**(2012-2016)**
Primary	: **Zapatera Elementary School**	**(2006-2012)**

PROFESSIONAL SKILLS

Language	: **C#, C, Java, HTML, CSS, Python**
Software	: **Adobe CC, Visual Studio, Unity, VS Code, MS Office**
OS	: **Windows**