December 2018

Personality AI Development

Jiuchuan Wang
Worcester Polytechnic Institute

Kien V. Nhan
Worcester Polytechnic Institute

Personality AI Development

A Major Qualifying Project Report
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science
In
Computer Science

By

Kien Nhan
Jiuchaun Wang

Project Advisor
Professor Gillian Smith

Abstract

This project is about exploring various methods to create game A.I.s with different personalities. A Japanese themed game Action Role Playing Game is also made for testing the A.I.s.

We used three algorithms: behavior tree, behavior tree with utility based decision maker, and finite state machine, and made four A.I.s each with personality of: aggressive, bullying, cowardly and tricky. All the A.I.s is put into an NPC monster called Nue in our game: Yokai World and a test were conducted with 16 play testers to see each personality's recognizability and believability.

Acknowledgement

We would like to thank our project coordinator, Jennifer deWinter for giving us the opportunity to do our MQP project in Japan. We would like to give our thanks to our project advisor, Gillian Smith for providing guidance throughout the time in Japan.

We would like to thank Noma Sensei, Ruck Sensei and Takada Sensei for taking care of us during our stay at Ritsumeikan University. We are very grateful for Takada Sensei for giving us a great workspace in his lab in addition to providing us resources to complete the project.

Table of Contents

Authorship

Chapter	Title	Authorship	Edit By
	Acknowledgement	Kien Nhan	Kien Nhan
	Abstract	Jiuchuan Wang	Jiuchuan Wang
	Table of Contents	Kien Nhan	Kien Nhan
	List Of Figures	Kien Nhan	Kien Nhan
	List Of Codes	Kien Nhan	Kien Nhan
1	Introduction	Jiuchuan Wang and Kien Nhan	Kien Nhan
2.1	Japanese Culture	Kien Nhan	Kien Nhan
2.2	Game AI	Kien Nhan	Kien Nhan
2.3	Personality AI	Kien Nhan	Kien Nhan
3.1	Experience Goal	Jiuchuan Wang	Jiuchuan Wang
3.2	Gameplay	Jiuchuan Wang	Jiuchuan Wang
3.3	Background Design	Jiuchuan Wang	Jiuchuan Wang
3.4	Player Design	Kien Nhan	Kien Nhan
3.5	Enemy Design	Kien Nhan	Kien Nhan
3.5.2	Personality: Aggressive	Kien Nhan	Kien Nhan
	Personality: Tricky	Jiuchuan Wang	Jiuchuan Wang
	Personality: Bullying	Kien Nhan	Kien Nhan
	Personality: Cowardly	Jiuchuan Wang	Jiuchuan Wang
4.1	Unity	Jiuchuan Wang	Jiuchuan Wang
4.2	C#	Jiuchuan Wang	Jiuchuan Wang
4.3	Spriter	Kien Nhan	Kien Nhan
4.4	UtilityAI	Kien Nhan	Kien Nhan
4.5	Panda BT	Kien Nhan	Kien Nhan

5.1	Implementation Intro	Kien Nhan	Kien Nhan
5.2	Class Design	Jiuchuan Wang	Jiuchuan Wang
5.3	Unity Technical Implementation	Jiuchuan Wang	Jiuchuan Wang
5.4.1	Aggressive	Kien Nhan	Kien Nhan
5.4.2	Tricky	Jiuchaun Wang	Jiuchaun Wang
5.4.3	Bullying	Kien Nhan	Kien Nhan
5.4.4	Cowardly	Jiuchuan Wang	Jiuchuan Wang
6.1	Overview	Kien Nhan	Kien Nhan
6.2	Population	Kien Nhan	Kien Nhan
6.3	Methodology	Kien Nhan	Kien Nhan
6.4	Survey	Kien Nhan	Kien Nhan
6.5	Challenges	Kien Nhan	Kien Nhan
6.6	Result and Analysis	Jiuchuan Wang	Jiuchuan Wang
7	Conclusion	Jiuchuan Wang	Jiuchuan Wang
8	Post Mortem Intro	Kien Nhan	Kien Nhan
8.1	What We Learned?	Kien Nhan	Kien Nhan
8.2	What Went Well?	Jiuchaun Wang	Jiuchuan Wang
8.3	What Went Wrong?	Jiuchuan Wang	Jiuchuan Wang
8.4	Future Development	Kien Nhan	Kien Nhan
	References	All	Kien Nhan
	Appendix	Jiuchuan Wang	Jiuchuan Wang

List of Figures

List of Codes

Chapter 1: Introduction

Artificial Intelligence (A.I.) is an important part of a game by bringing Non-Player Characters (NPC) and enemies to life. It allows NPCs to act intelligently, and able to challenge, impress, or even bring emotional attachment to the player. It has become a vital part of many games, as the gaming industry is evolves. Graphics have been getting better in last decade and game has been moving towards an open world genre. An open world environment allows player to craft their own story. With the emergence of virtual reality, immersion within the game world has become a more important than ever [28]. AI has an important role in interacting with the players; they are part of the player's story and their involvement can either make the game better or make the game stagnant.

In our experience when dealing with an A.I. enemy for Action Role Playing Game (ARPG), we are usually required to figure out patterns for its attack and movement, so we can act accordingly and exploit its weaknesses. During this process, the personality of this enemy character and preferences on certain actions of its A.I. are gradually revealed to us, the players. Different A.I. behaviors and personalities give players different attitudes and approaches toward them. It makes the game engaging.

We were interested in working with AI for the project because we wanted to learn more about it and it was perfect for a team of all CS majors. Influenced by our research we decided that the direction of our project was personality AI. In this project, we made a game related with Japanese culture, explored different ways to create A.I.s with personality, and make various A.I.s with different personalities using different techniques for our game. Due to our team limited experience working with AI, we turned our attention to other works done by researchers to learn about the approaches to creating a personality AI. We used two approaches from them. One was a brute force approach that requires planning out different scenario for the AI and another was allowing the AI to make its own decision in a less scripted method.

Our game *Yokai World* is a 2D action game in which the player is trapped in an alternative world of Yokai. In *Yokai World*, players must defeat the Yokai trying to kill them. Players must play through the game four times, learning the different personalities of the Yokai. We learned about the effectiveness of our AI by getting input from the players through a session of play testing our game.

In this paper we talk about the overall development of our game. In chapter 2 we talk about the background research that inspires the designs and gameplay of our games. In chapter 3, we explain the designs, the features and the gameplay that is in our game. In chapter 4 is about the software and tool we use in our project. Chapter 5 is about the AI and Unity implementation. In chapter 6, we discuss in details of our play testing and the analysis. In chapter 7, we talk about the conclusion of our results. Finally the last chapter is the postmortem of the project. We talk about the highs and lows of our projects in both the technical and the non-technical of the project such as: what we learned, what went wrong, what went right, and what can be done for future works.

Chapter 2: Project Background

Before designing *Yokai World,* we conducted background research on three main areas of our game, Japanese culture, Game Artificial Intelligence, and Personality AI. The aspect of the Japanese culture we explored were the Japanese Folktales of Yokais and Japanese Temples and Shrines .Then we discussed briefly about the Game Artificial Intelligence in games and the algorithms that games used in gaming industries. Finally we explained the importance of personality AI to our project.

2.1 Japanese Culture

2.1.1 Yokai

Yokais are supernatural beings usually depicted as ghosts, monsters, demon, and mostly deformed animals or human. They are mostly conjured up from either ancient folklores, or imagination of the writers or storytellers during the Edo Period (1603-1868) [52]. Yokai was used as concept that explains unscientific phenomenon. People once believed that natural disaster like, lightning, tsunami, earthquakes and forest fire were caused by Yokais who wanted to terrorize people. However not all Yokais were considered evil; there were good Yokais well [50]. There are no record found that explains their origin but when it comes to a database of the type of Yokais told in history, there are a lot of them. Yokais were not popular until the Edo era, the time of art and culture. The concept was used as a way to tell interesting stories. It later embeds itself as part of Japanese art and cultures as more and more stories are adapted to today media.

Toriyama Sekien was a scholar and artist who traveled the country and documented stories of yokai in his encyclopedia. His famous works were *Gazu Hyakki Yagyo* (The Illustrated Demon's Night Parade) and *Hyakki Tsurezure Bukuro* (A Horde of Haunted Housewares) [22]. They are encyclopedias of known Yokai heard across Japan.

Mizuki Shigeru was a manga author and artist who preserved these Yokai stories by compiling them into manga series. His famous work was Ge Ge Ge no Kitaro, a children manga series. His effort inspired others to continue the traditions by recreating the Yokai stories by adapting it into many different media format [50]. As a result these Yokai stories continue to exist to the present day.

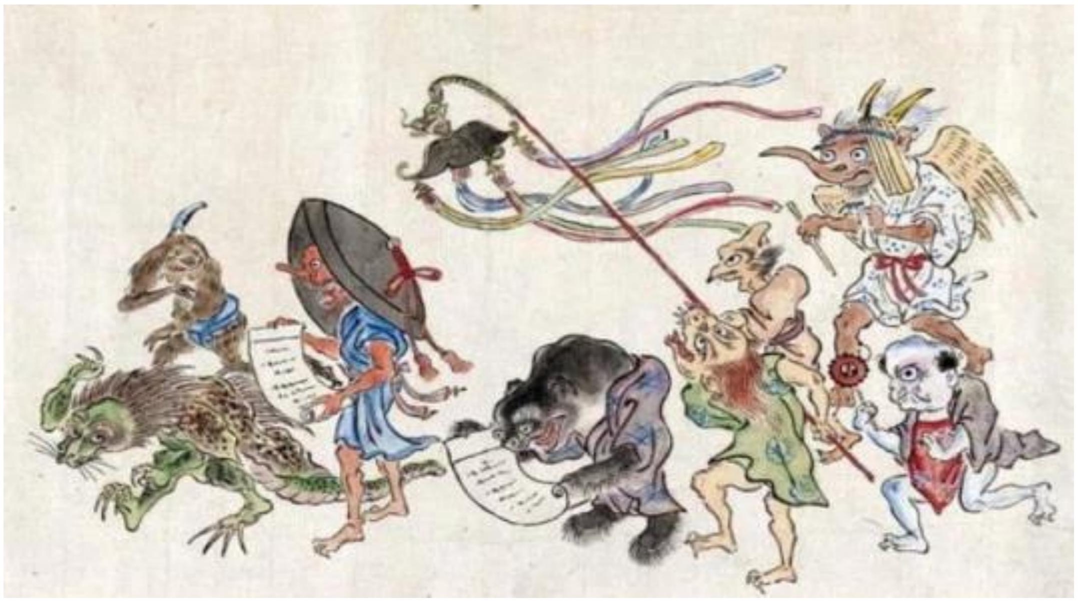


Figure 1: A portion of the Night Parade of the 1000 Demons Painting [50]

You can also find that today Yokai appears in the current media. For instance the popular game *Yo-kai Watch* uses Yokais from stories and legends as a theme of the game. Other games include *Dark Souls, Pokemon, Shin Megami Tensei, Persona and Nioh*. It not only limited to video games, but also anime, manga, TV shows and other popular form of media as well. Majority of Japan population is very familiar with the concept of Yokai because it has been preserved so well throughout history. It is no surprise that the concept is so popular in Japan because generations of children, young adults, and older individuals grew up with it.

Because of its deep connection to the Japanese Culture, we decided to use Yokai as a theme for our game. There are multiples Yokai to choose from but since our project center is in Kyoto, we decided to use a Yokai named Nue, who had a history in Kyoto. In our action game, the player battles Nue, an evil monster who terrorized people in the legends. In addition, we developed four different personalities for our Nue. Each personality is inspired by stories of Nue and other stories. More details of the personalities are described in section 3.5.

Nue is well-known for a particular story told in the Heian era. During the Heian period, emperor Konoe had nightmares every night and eventually grew very ill. Nothing could be done to cure him. It was discovered that the source of his illness was an effect of a Nue so a legendary samurai, Minamoto no Yorimasa, was summoned to kill it. Once it was killed, emperor Konoe recovered completely. Due to the fear of being cursed, its body was set to drift down Kamo River. Eventually, it reached a shore in Hyogo where villagers gave it a burial mound to avoid being cursed [30]. The burial still continue to exist even today in Matsuhama Park in Hyogo under the name Nuezuka.

`

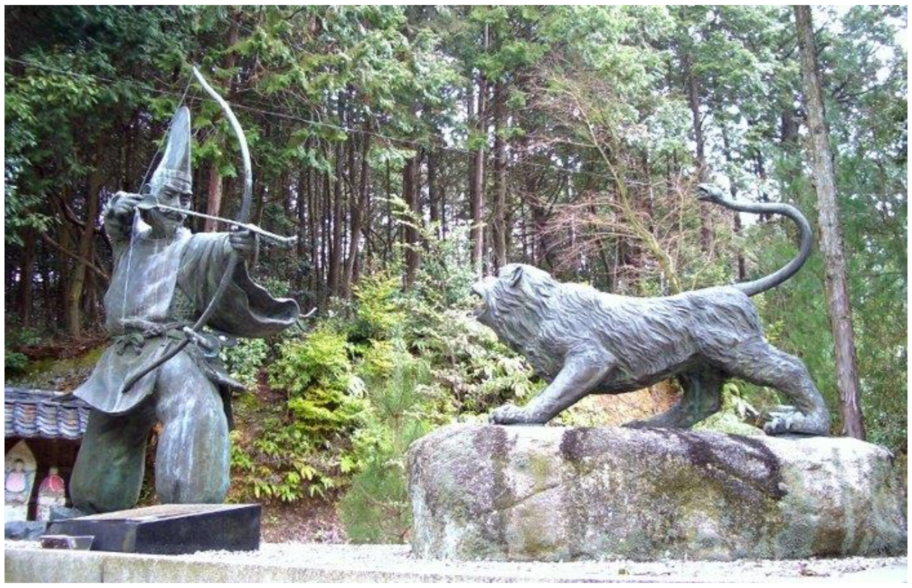


Figure 2: Image of Nue and Minamoto no Yorimasa Memorialized in Matsuhama Park in Hyogo [12]

Due to the popularity of its existence in Japanese culture, Nue has been adapted in many media form. In the critically acclaimed game *Nioh*, Nue is a semi-boss that the player must defeat to move on. Another game series that Nue makes an appearance is the Shin Megami Tensei, where Nue is a potential enemy to defeat. In *Okami*, there is a monster that is called a chimera, but although its appearance differ a bit from other games mentioned, it was said to be based on Nue.

Other than video games, Nue makes appearance in anime and manga. In the anime Mononoke, Nue is depicted as a spirit of a rotting wood that gives people great power. It enjoys killing people who seek its power [8]. Nue also appears in popular anime of recent years such as *Boruto, Bleach*, and *Nura: Rise of the Yokai Clan*.

These adaptations of Nue in media make it out to be a more complicated character than the legend suggested. Although a beast, Nue is given human like emotions and characteristics, suggesting that Nue is more than just a simple beast. Because of its status as a Yokai, or evil spirit, Nue has many shape and form in these popular media.

2.1.2 Shinto Shrine and Temple

Having a setting and background design for our game is important so we needed to research on the appropriate location for the theme of our game. When you see a Yokai-theme anime/manga, you usually see it in a temple and/or shrine. One of the religions of Japan is Shintoism. Shintoism is a concept of animism and worship of nature. It is commonly practiced at a Shinto Shrine [4]. Everything was thought to have a spirit within it. Some spirits were worshipped for good fortune while others were scorned for their misfortune; the concept of kami and Yokai are sometimes interconnected. The two supernatural concepts of Shintoism and Yokai meshes together very well, which is why you see them together often in anime/manga where supernatural is the theme. Anime/manga like *Nura: Rise of the Yokai Clan*, *Natsume Yuujinchou*, *Uchouten Kazoku*, and *Ao no Exorcist* are anime that features Yokai and each have mentions of shrines and temple as it's setting [3]. Because most of the members in our group are anime watcher, we linked Yokai and temple/shrine unconsciously. Not only anime but we can the two together in games, TV shows, advertisements, and even inside shrines and temples that we visited. Being in Japan, we wanted temple and shrines to be the setting for our game.

We had the pleasure of visiting a temple in the Kyoto. We visited Entokuin, a sub-temple of the Kodaiji Temple. It was built by Kitano Mandokoro or Nene, the wife of Toyotomi Hideyoshi, as a way to mourn for her late husband. Kodai-ji Temple is located in the Higashiyama Mountains of eastern Kyoto. It was fortunate that Kodai-ji Temple buildings survived a fire in 1789 and remain to be in excellent condition to this day. It is considered an Important Cultural Property by the Japanese Government [1].

Entokuin had a very calm and soothing atmosphere which reflects the temple's purpose. The tatami matted rooms have little to nothing but has so much characters embed in its design. Each room is divided by sliding doors, which has intricate designs on them each carry different meanings. One sliding door was painted with pine tree, bamboos, and plum-blossom. They are considered to be 3 strongest trees to survive the winter. It is used in Japanese garden as a symbol of being strong throughout adversary.

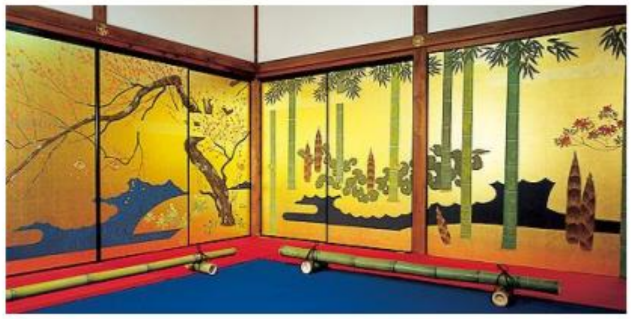


Figure 3: "Syoutikubai" (pine tree and bamboo and plum-blossom) Syoutikubai means good luck [11]

Inside Entokuin, there were displays of Night Parade of the 1000 Demons. Unfortunately, we were not allowed to take photographs of the art piece. It looked similar to Figure 1 except it was a section with different Yokais, but similar art style.

You can find folklores of Yokai such as Tengu, Kitsune, Oni, and other known Yokai who had affiliation with temples/shrines. For example, in the story of a famous samurai, Minamoto no Yoshitsune, he was trained by a tengu king Sojobo at Kurama Temple, who helped him become the greatest swordsman to ever lived. Although a Yokai, Tengus are considered noble in modern Japan society [40]. You can find statues of them in temples and shrines, signifying importance to the culture and the location. Another instance is on February 3, a widely celebrated festival named Setsubun has a ritual in which people throw beans out from their home or at someone dressing up as an Oni while chanting "Oni wa soto, fuku wa uchi" which translate to "Oni outside, fortune inside". The bean throwing tradition first emerged in the Muromachi period (1337 – 1573). It was used to purify the home by driving away evil spirits in order to bring good fortune to one's home [39]. Because of the connection, we decided it was worth having temple/shrine as the setting for our Yokai-theme action game.

2.2 Game Artificial Intelligence

The existence of Artificial Intelligence in games allow developers to create a unique and compelling experience for players. Game developers view AI as the future of gaming because as technologies advances, more engaging gameplay and environment are desired with AI being an important piece in getting the player involved and invested [34].

Based on the readings, having a human-like AI in one's game is very desirable, in theory. But in practice, it all depends on the design of the game.

There are three factors that make up a successful AI. The AI has to be responsive, the AI should be challenging, and the AI should be fair [26]. An AI has to be able to give good feedback to the player. It means that if the player shoots, the AI should dodge and if player is open, the smart thing for the AI is to shoot at the player. The AI has to be alive and moving to match the player's action. A challenging AI does not have to be unbeatable, it just needs to give the player a tough time and make the player work for the victory. In addition, the difficulty has to adapt to the player growth as they play the game. A fair AI means that it should not have access to information that the player does not have and it should not be again, unbeatable as a design. The AI should be on in a sense, on an equal playing field, no advantages that cannot be exploited.

According to a study, it was found that people enjoyed playing against other human because human players are "intelligent, unpredictable and adaptive" [37]. A human-like AI has to be able learn and exhibits believable behavior [29]. The approach that usually goes with this concept is machine learning. While machine learning sounds nice, game industries do not use it because it is too unpredictable and make controlling the gaming experience even harder [41]. In addition there must be a large amount of training data the

AI needs to learn from in order to sufficiently function. Not to mention the resources that is needed for this to be successful.

There is also another group from the same study that prefers to play against computer because it was more convenient [37]. The study mentioned showed that people play games for different reasons. There are many different people in the world. Some plays game to escape reality, some play games for the thrill, and some play games because they needed a distraction. What is considered a “fun and engaging game” is subjective and very difficult to gauge. Overall the desirability of achieving human-like AI depends on the design goals of the game and who the audience is.

Developers would only create an illusion of intelligence in their game AI; just enough for players to acknowledge that the AI is behaving logically and relative to what they know. In our game, we aimed to show behaviors that are clear indicators of each personality. We wanted to ensure that the behavior we chose for each personality is logical and reasonable.

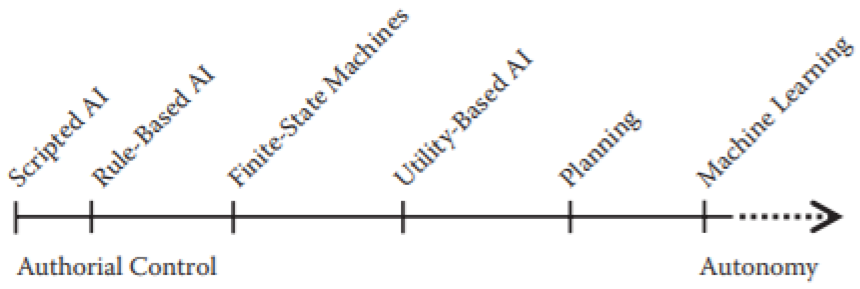


Figure 4: Types of AI Architecture and Developer Control [41]

There are two general techniques game industries employ to make their AI believable. One strategy is the animation. Simple actions like character moving faster indicate a high emotion such as anger, agitation, nervous and upset. While slow movement indicates calmness. Having the AI display these actions in line with the situation can be powerful in projecting intelligence because human acts similarly to those emotions a well. Simple animation practice includes having the anticipation before performing an action and body moving even when standing still [42]. All for the sake of enticing the player the realism of the AI and the game world. While this technique is useful and a standard in game development, we cannot use this technique in our game because none of the team members have experience with animations or have the ability to accomplish this.

Another strategy is to use AI architectures to create the AI. This technique follows a coding approach that all members of the team are familiar with and so we knew that this was the approach to take in developing our AI. Architectures include scripted AI, Finite State Machine, Utility-Based AI, Planning and Machine Learning. Developers want the AI to be reactive and adapt to any situation. For the AI to be adaptive, it needs to be able to react to changes, from player’s action and the game environment [41]. This give the developers some control over the player experience in deciding how the AI reacts to different situations. There is no right answer to how an AI is created. All games can succeed by using different type of AI architecture. For examples, games like The Sims relied heavily on AI use architecture like Utility-Based.

2.2.1 Algorithms

Behavior Tree

Behavior Tree is a tree that consists of nodes that an agent use for decision making. A behavior tree starts at a root node which has many children as behaviors. Each behavior has a condition that must be met before executing. Each behavior is a node. It can execute as a sequence node, traversing the children in order and the node only fails when one child fails. Or it can be executed as a selector node where if one fails, the algorithm moves on to the next child and if all children fail, the parent node fails [41].

The tree can be as straightforward as it needs to be, meaning you can implement a behavior tree with many different behaviors each with only one child or it can be as complicated as you need it to be with any number of branches. Behavior tree is stateless, meaning it does not need to access any memory to know what behavior to execute. Each behavior in a behavior tree is independent of each other so adding and removing a behavior will not affect others. A drawback from behavior tree is that the algorithm can get really slow when there are too many behaviors to process. Decision making is slow and it can be limited by its design. The reason is that tree will process each behavior in sequence so if a behavior that can only be executed is at the end of the list of children, it can take up a lot of processing time. However the issue can be avoided by having a smart behavior tree design.

This is a standard for game AI so we figured it was a good starting point to learn how game AI works and what we can do with it in our game. This architecture became the implementation for one of four personalities described in chapter 5.

Utility-Based

In simple terms, a Utility-Based AI makes decisions based on the score calculated from some mathematical formula and select the action with the highest score [41]. Utility-Based AI answers the question “How much I want ……” rather than simply deciding what action to take [42]. It takes into account of the weights of its need, the rank, and any considerations that needs to factor into the decision process before deciding on the best action. This architecture is used in situations where there are numerous actions to choose but there are not right actions to take, just the most logical based on the score.

As the game situation changes, the scores for most of the actions will change. This makes the AI appears more autonomous and less predictably scripted. The system allows the AI to truly adapt to the changes and makes it interesting to play with or against. The disadvantage of the Utility-Based is finding the “right” formula for each action. Because the score of the actions are calculated by a formula, having the “wrong” formula can make the agent act unpredictably. It does make the agent appear more natural rather than robotic; however, sometimes you want the agent to act correctly to specific situation instead.

A well-known game that uses utility is the Sims. In order to be able to model basic human behavior, there are needs that each sim possess and each of those needs affects how the character acts in different situation. In the Sims, the sim can live and make decision without the player actually controlling the sim [51]. The algorithm in the sim chooses the four best choice for the sim to choose from. For instance, if the sim is low in energy, they would sleep to avoid being cranky. If the sim is hungry it will eat to avoid being upset. If the sim is bored, it will entertain itself. The list goes on.

The way that utility is used in the Sims was very interesting. It was a perfect example of how to model human behavior in an AI, similar to our project. We decided it was worth attempting to implement it in our game to model personalities.

Behavior Tree with Utility Based

Behavior tree is a common AI architecture used in game industries. It is simple design that allows developer to map out how they want their agent to perform. The huge issue mentioned earlier with the Behavior tree is that it does not specify which actions it should do without traversing the all condition of the tree. In a behavior tree, each action is checked one by one in the tree until it reaches the end. In the Figure 5, this shows a typical behavior tree with options for an agent. As it stands, the leftmost node will be processed first and will either move on to the next node if it failed or return the result back to the parent. The design lacks the ability to make a decision without fulfilling the conditions specified.

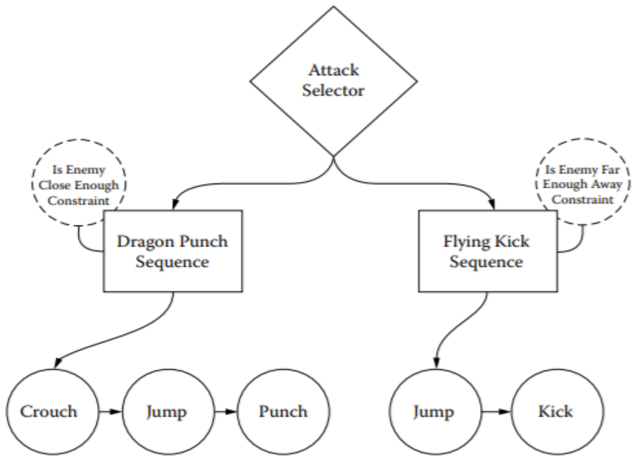


Figure 5: Sample of A Behavior Tree [19]

To address this huge negative, a utility-based system is used together with the behavior tree. Utility-based system allows the agent to make a decision by choosing the highest utility score of all actions. The utility score should be used as a selector in order to mask the weakness of the behavior tree. This helps make our agent more spontaneous and less rigid in its execution.

In the Figure 6, the AI is trying to decide which action sequence to take. It can seek medic if health is low, shoot, or seek medic again but this time comes with another condition, when monster is out of sight. There is duplication because you have multiple conditions that utilizes the same action, but at different instances. Choosing medic first ensures that the AI will prioritize healing. If the first fails, then it will shoot, however if it cannot shoot then it will heal again but this time with a different situation; notice that the developer have to plan out each different scenario.

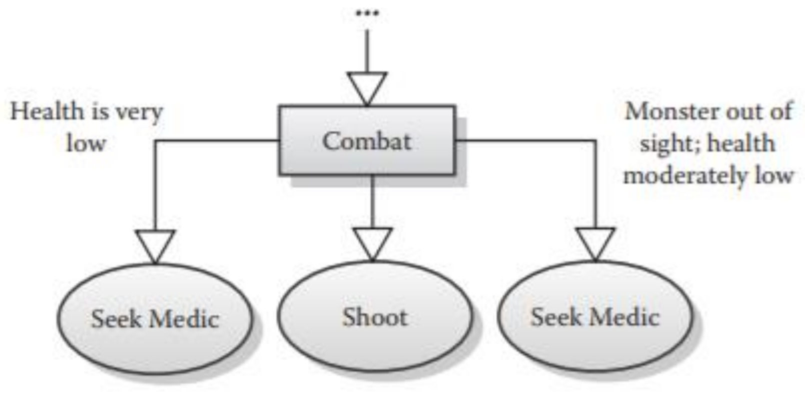


Figure 6: Sample of Behavior Tree with Duplicate Action [15]

Figure 7 address the issue by having the Combat node to calculate the utility score of all three possible actions and choose one that has the highest score, thus the most logical choice for the situation. This eliminates the need for having duplicated actions that requires different condition to activate.

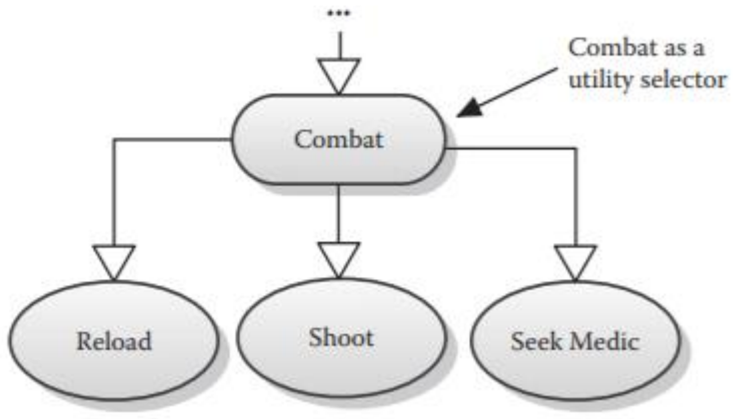


Figure 7: Sample of Behavior Tree with Combat as Utility as selector [15]

This combination of behavior tree and utility looked very promising. It complemented well with each other by addressing the weaknesses that each possess. We decided to use this technique to develop the personalities for the AI because we never really thought about combining multiple architectures into one and we are interested in learning how this technique can make our AI better or worse in its autonomy.

Finite State Machine

Finite State Machine is an algorithm that simulates an agent action using states. A state, in simple term, is a status of the agent at a given time. For instance, an AI agent can be in Idle State, where it is just standing around or Running State, where it is currently running. State answers the question, “What the Agent is Doing/Thinking Right Now?” The FSM can transition to another state or it can continue to stay in one state. Transition between states requires the agent to fulfill certain conditions [42]. For instance, in order to transition to Running state, it must check the condition that the player is within its sight radius, if the player is in the radius, the agent will now be in Running State and it will perform a running action, if not, the agent will remain in the Idle State or even transition to a different one. The process repeats every frame. This architecture organizes the logics.

State Machine is very basic and it has a small learning curve. Like behavior tree, it was used as a way to learn about decision logic that comes with developing an AI. Since we are new to the game development process, we are not sure which architecture to use so state machine is one of several algorithm described that we wanted to explore.

After looking into the algorithms that games uses, the next step was to figure out how to model the personalities.

2.3 Personality AI

When playing a game, it is very easy for players to recognize whether the NPCs they interact with are satisfying or not [43]. They see patterns and repetitions. It can negatively affect the player experience when player’s action has no real impact on the game environment. A way to address this is add realism to the NPCs by giving them personalities [25]. Giving an AI personalities add a new depth to it, making it a more interesting character to interact with. One approach to achieve this is to consider all possible scenarios that can occur from player choices and incorporate those into the AI. The realism that the AI can project from this approach depends entirely on the scenarios that developers can think of. Another is to give the AI a little bit of autonomy and allow it to make decision themselves. One study suggested that it is more worthwhile to aim for credibility instead of realism because you have more control [29]. The idea of credibility is consistency. For example, in a game of NBA2K18, a seven foot center should dunk majority of the time when he is under the basket just like in real life. Or a player with an aggressive personality should attack the basket more often than a player with laid back personality. It does not have to be human-like, it just need to behave consistently. Our approach to developing our AI was similar to these two approaches. Using the algorithms we learned from previous sections, we were able to utilize them in these approaches. And we made our AI behave consistently to the personality that we assigned it to.

One personality model that was mentioned several times in the readings was the Big Five Personality Model. Modeling realistic behavior in AI requires background knowledge about personalities. The Big Five Model offers a guide to understanding what personalities are. The Big Five are OCEAN, openness, conscientiousness, extraversion, agreeableness, and neuroticism. All five factors makes up of a personality. The general consensus in developing personalities was to give the AI multiple traits that can change any time thorough gameplay. These traits are created with the Big Five Factors in mind.

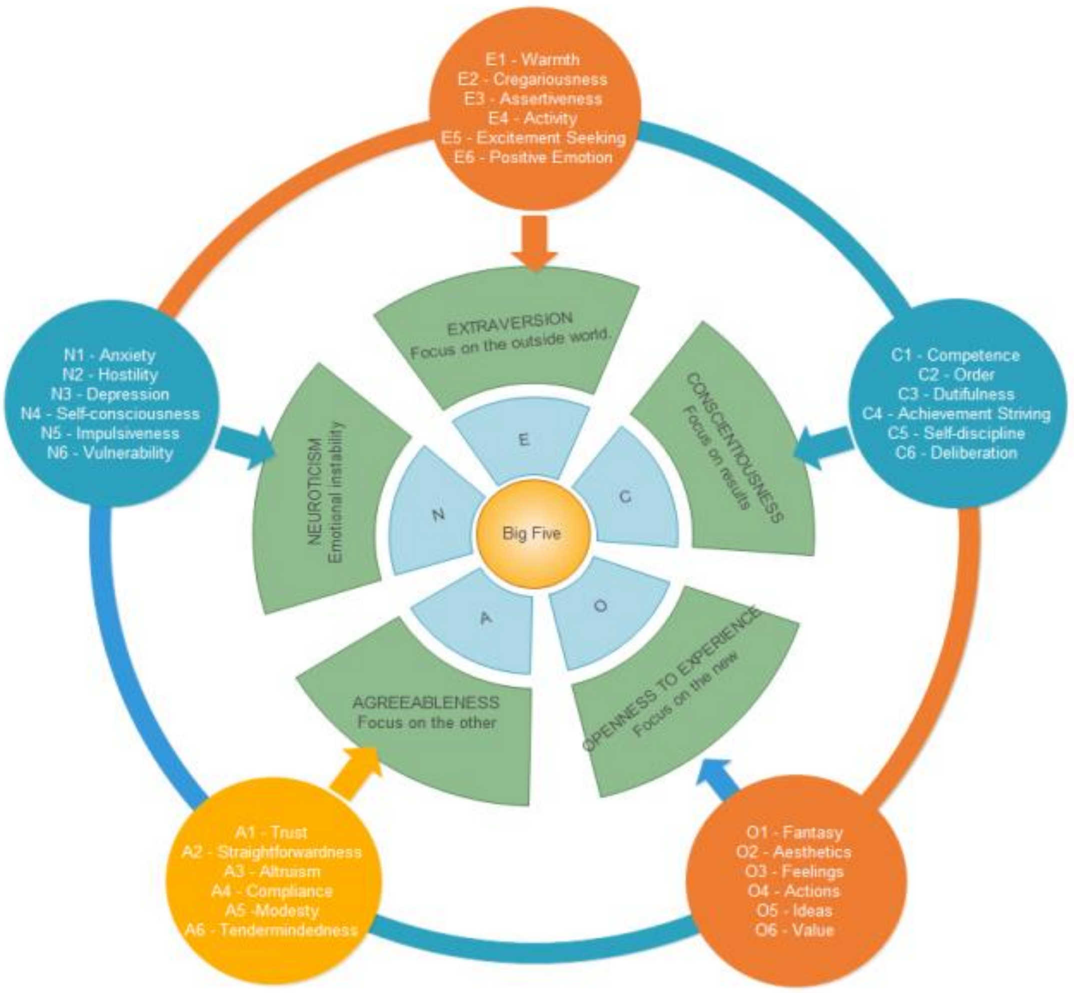


Figure 8: Big Five Personality Circle [7]

We model our personality by showing different range of actions in different situations that characterizes our AI. For instance when our AI is at low health, the AI would either run at player or run away. If it runs away, it shows a hint of cowardice, nervousness, or disorderly. If it runs at the player, it shows that the AI is assertiveness, fearless, and/or predatory. The string of action our AI takes throughout the gameplay says something about its personality. This was the approach we took in displaying the personalities of our AI. Our rationale for each personality is described in section 3.5.

Chapter 3: Design & Gameplay

3.1 Experience Goal

3.1.1 Different Experience with Different AI Personalities

The most important part of our experience goal is to let the player notice differences between different personalities of AI and enjoy playing with them. Ideally, the player should be able to tell which personality is which after playing with it for one round and give a justification for their decision.

In order to make it easier for the player to feel the difference between the personalities, we built multiple behaviors for the AI, such as attacking with fist, charging attack and range attack. Different personality would have different choice of behavior, and they would also have different reaction to the player’s behavior. For example, when the player goes near the monster to attack, the monster will try to run away if it is a timid AI, while it will attack the player if it is an aggressive one. If it is a bully AI, it may use charging attack all the time. However, if it is an aggressive AI, it will be more likely to chase the player around and attack him with its fist. More detailed of the personalities of the Nue are explained in the Section 3.5.

We also created plenty of animations and special effects, including sound and visual ones, to make the monster’s behaviors vastly different from each other. This will help the players to tell if the monster is acting differently and make it easier for the player to play the game, since vision and sounds are the only ways for the player to receive information from the game.

3.1.2 Multiple Strategies to Win

We chose to let the player have multiple strategies to play through the game, because we didn’t want to limit the player’s options when they fight with the monster that has different personalities. The map is basically a large room where the player can walk up, down, left, right and diagonally. Although the player can only by waving his sword, there is another special ability of player called “Dodge”, which lets him move faster on the map and dodge the enemy’s attack. In many games that are made by RPG-Maker, player could get stuck by a little obstacle on the map, while this game allows the player to go anywhere they want on the map and the players are not restricted by the map tiles.

All these features give players more options to fight the monster. They can create their own strategy according to different personalities of the enemy.

3.1.3 Cultural Experience

All the visual effects, sound effects and art design in this game help to build an Japanese cultural environment. We went to temples in Kyoto, including Kodai-ji Temple and museums which contains historical Japanese drawings.

Players should be able to tell that the background is a Japanese-style room, and the monster should be easily known as an eastern monster, not something like a vampire or werewolf. We don’t want to add too many details into the environment, because we are afraid that the environment with too many details will attract players and divert their attention from the difference between personalities of the AI.

Each A.I. should be believable and fit the general impression or historical reference of the monster.

3.2 Gameplay

Our game is a top-down 2D fighting game. A fighting game is a video game genre based around combat between a limited amount of characters, in which they fight until they defeat their opponents or the time expires. For our game, it’s player versus AI and doesn’t have any time limitation; therefore, the player has to beat the AI to win the game. The fights consist of several rounds and take place in a small room. Player and monster will have different abilities. In this game, the player can move, dodge and attack.

The gameplay refers to a few famous indie games. They are all top-down 2D action games. The reason that we chose this gameplay is, as we mentioned before, to give player more options on playing the game and not limiting them. They can go anywhere they want on the map to dodge the attack from the enemy and attack them back.

3.2.1 Wizard of Legend

Wizard legend is an indie roguelike video game released by American studio Contingent99 and Humble Bundle in May 2018 [10]. Players act as wizards as they play through a procedurally-generated dungeon, and they can defeat mobs, three major bosses, and a final boss with a wide variety of spells to become “Wizard of Legend” in the end.

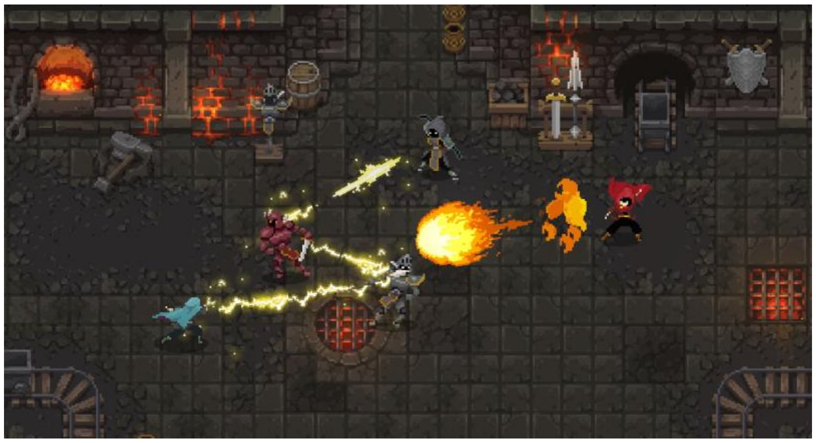


Figure 9: A Snapshot of Wizard of Legend [46]

In our game, the player will not be able to shoot projectiles, but we implement another mechanic from *Wizard of Legend*, which is dodging. In *Wizard of Legend*, players are allowed to cast a spell to dash to a certain distance. Players of our game can dodge by pressing the space button, and then they can dash to a little distance to dodge the attack from their enemy.

We also learnt to implement the moving and camera mechanics from *Wizard of Legend*. Since our game is a top-down 2D game, just like *Wizard of Legend*, the moving controller of the player is by keyboard keys - W, A, S and D. In this game, the camera not only follows the player but also tries to keep every enemy inside of the camera so that players know where their enemies are, and we implemented this mechanic as well.

3.2.2 Binding of Isaac

The Binding of Isaac is an indie rogue like video game designed by Edmund McMillen and Florian Himsl [16]. Players control Isaac or one of the other characters through a procedurally generated rogue like dungeon. Players need to kill monsters while collecting items and power-ups to defeat bosses, including Isaac's mother.

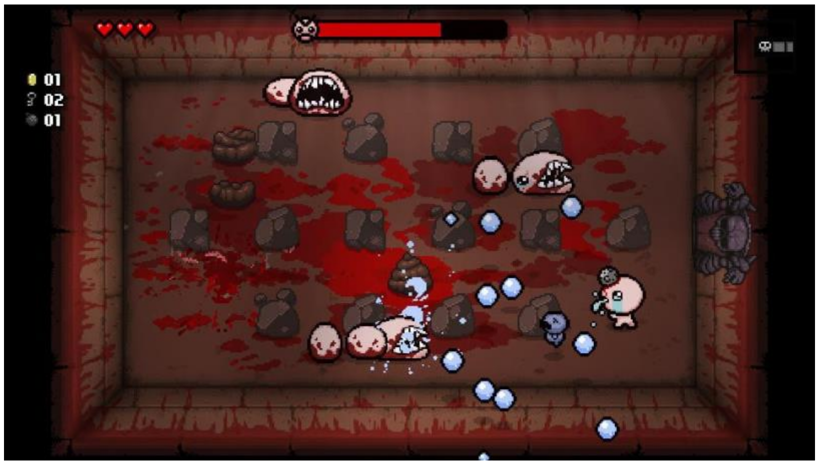


Figure 10: A Snapshot of Binding of Isaac [33]

One thing we learnt from this game is that we need to add visual effects to the game where players hit an enemy or they get hit. In *Binding of Isaac*, when monsters or players get hit, they will shine in a red color for a few times so that players will notice that they get hit or hit their enemies. Usually this effect will last for half a second. We designed two different visual effects, which show up when players get hit or hit the monster. The one for players hitting the monster is a reddish huge slash that shows up on the monster's body, which is indicating that the monster was damaged by player's sword. The other one for monster hitting the player is a yellow huge strike animation that plays on top of the player's body. This indicates that the player was hit by the monster and he will receive damage.

Another feature we learnt from Binding of Isaac is its art design. The characters in this game have a big-head and small-body structure. Our art design of the protagonist refers to this art style because we think this art style is easy to manage and it would go well with our goal of keeping a sense of Japanese culture which is the manga style.

3.2.3 Player and Enemy Controllers

Player Mechanics

For the player, there are two main attributes and three main abilities. The two attributes are health and stamina. In this game, health is an attribute assigned to players and enemy that indicates how much energy they left to continue the game. Usually health is interpreted as hit point (HP) and players and enemy will lose certain amount of hit point when they are attack by someone else. When players run out of health, they lose.

The second attribute is stamina. Stamina is an attribute assigned to player's character which indicates how much energy they left to cast abilities or make actions. Every action made by the player, except simply moving, will cost a certain amount of stamina. Stamina will be recovered by 5% every 2 seconds. Actions made by players will not interrupt the recovery of stamina. This limited stamina mechanic could effectively stop player from spamming. We hope players can develop certain strategies against the monster, not just spamming keyboard to kill their enemy.

In our design, the player should be able to move vertically, horizontally and diagonally. The reason of this is that we want the player to have more ways to attack the monster and to dodge the monster's attack. This also offers the player more freedom on their movements. The player has a lower health than the monster, since monster "Nue", in our background research, is stronger than a human. In addition, in most of the fighting games, player usually has a lower health as well, since they are highly intellectual comparing to the AI of bosses. In our game, the AI is already pretty strong; however, the monster is stilled designed with more health. The reason is that we want players to have a longer game experience and we hope the monster could fight with the player for a longer period of time so that it's easier for players to tell the difference between each AI personality.

The player's movement speed, however, is way faster than the monster's speed. This is not only because player has a lower health, but also because we want to offer a more intense and exciting game experience for players. Movement speed is a statistic that represents the rate at which a character travels across the map. Faster movement speed requires the player to react at a higher pace. According to Matthew W.G. Dye: "playing action video games—contemporary examples include God of War, Halo, Unreal Tournament, Grand Theft Auto, and Call of Duty—requires rapid processing of sensory information and prompt action, forcing players to make decisions and execute responses at a far greater pace than is typical in everyday life", and "fast decisions typically mean more mistakes". [31] It seems like player is over-powered by this fast speed, however, they need to learn to control this speed otherwise they are likely to keep making mistakes during the fight. This also means that we successfully created another learning experience for players during the gameplay – controlling their movement.

An import ability of player is dodging. Players can dodge through a small distance in one direction instantly, and this will cost them 5% of their stamina. This ability can only be cast while player is moving. This looks like another buff for the player, however, they need to learn to use this ability as well. As we mentioned above, abilities cast by players will cost them stamina, including attack. This means the more players use their dodging ability, the less attack actions they can make to the monster. They need to think carefully before they move.

Attacking is another ability of the player. There is only one way for them to attack, which is waving their sword in one direction to chop their enemy. It can be cast only vertically or horizontally, and it is a melee attack so that players need to get close to monster, walking into its attacking range to hit it. This melee attack also adds another learning process to our gameplay – players need to learn how to deal damage without being attacked by the monster. They need to combine dodging and their melee attack together to fight with it.

The last ability of player is healing. Note that before we get our game tested by others, we decided to give healing items to players in the beginning of each round, because it’s too difficult, even for us, to win the game with such small amount of health. This healing item can recover all the health (hit points) of the player, and players are given 3 of them in the beginning of each round. Here comes another thing for them to learn – when should they use the healing item? If they cast it immediately after being attacked, it’s a waste of healing item because it’s able to recover all of the hit points. If players wait until there are only few hit points left, they could get killed by the monster before they press the key to use healing item.

Enemy Mechanics

Just like players, enemy can move vertically, horizontally and diagonally. Its moving speed is around 25% of players’ moving speed. This, however, does not mean that it’s going to moving slowly around the map all the time. When it cast one of its abilities, it moves faster than player.

Enemy controller only has one attribute and two abilities. The only attribute it has is health. As mentioned above, in this game health is an attribute assigned to enemies to indicate how much of its energy is left to continue the game. It does not have any ability to recover health because its health (hit point) is already enough – four times as much as the player’s health.

It doesn’t have stamina attribute; however, this does not mean that there’s no limit for it to use its abilities. There is a cool-down mechanic for all of its abilities. In this game, cool-down is a period of time that the monster has to wait for before it can cast its certain ability again. Both of his abilities are restricted by cool-down so that it cannot keep attacking the player and this gives player a chance to attack the monster as well.

One of its abilities is punching. By punching he waves his fist in one direction and deal damage to players if he hits them. As its description, this attacking ability is a melee attack. Monster needs to go close to players to attack them. There is a 1 to 2 seconds cool-down for using this ability. Punching will not deal lot damage to the player, since the cool-down of it is short.

The other ability is charging. Charging action can be divided into two stages. The first stage is charge up. Charge up means they need to stay at the same position for around 1 second to gather its strength. This stage also reminds the player that it’s going to charge on them, because the monster is shaking its body while charging up. In the end of first stage, the monster also decides in which direction he is going to charge. The second stage is charge out. Charge out means that the monster could dash in a certain direction until it hits the player or the wall. He can charge in any direction, including diagonally, while he cannot change its charging direction while he is charging out. Charging attack will deal a lot of damage to players if they were hit by it, since it’s rather difficult to hit the player. Even if the moving speed of monster is faster than player’s moving speed while it’s charging out, it’s still hard to hit the player because players can dodge charging attack with their dodging ability.

3.3 Background Design

3.3.1 Cultural Elements:

In our game, we tried to keep it obvious that the player is fighting in a Japanese house. At least they should be able to notice the Japanese style in this room. Therefore, we went to several places and search on the internet to find resources to design backgrounds of the game.

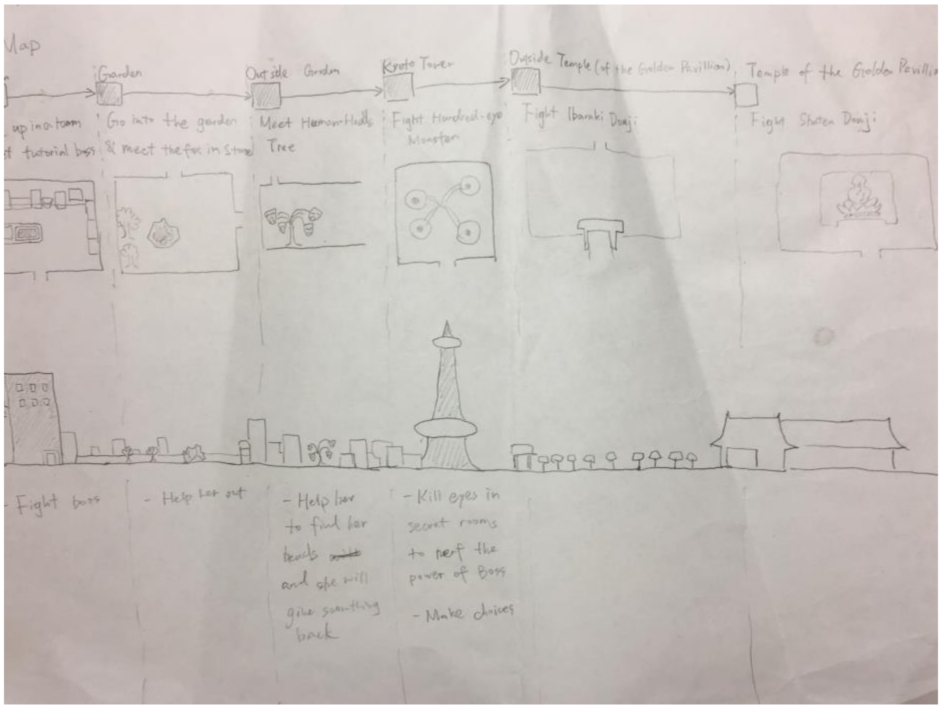

Map
Garden
Outside Garden
Kyoto Tower
Outside Temple (of the Golden Pavillion)
Temple of the Golden Pavillio
up in a town
tutorial boss
Go into the garden
& meet the fox in Stone
Meet Human-Heads Tree
Fight Hundred-eye Monster
Fight Ibaraki Douji
Fight Shuten Douji
- Fight boss
- Help her out
- Help her to find her heads ~~with~~ and she will give something back
- Kill eyes in secret rooms to nerf the power of Boss
- Make choices

Figure 11: Original Design of Map

3.3.2 Temple

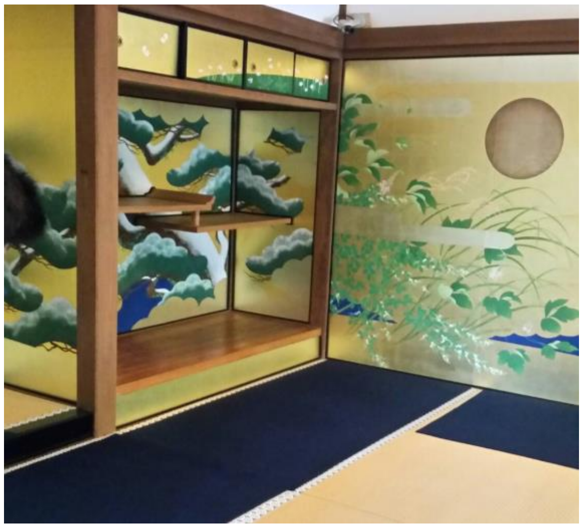


Figure 12: A room in a Japanese temple[1]

We went to several temples in Kyoto and searched for pictures of temples online. We noticed that in most of the rooms the floor was made with tatami. Tatami is a traditional Japanese flooring material. If we put tatami into the background, the player could notice the Japanese cultural environment easily; however, in most of the temples, tatami is used only in bedrooms and the floor of other rooms are wooden. Since our game took place in a large living room instead of a bedroom, we chose to design a wooden floor for the background.

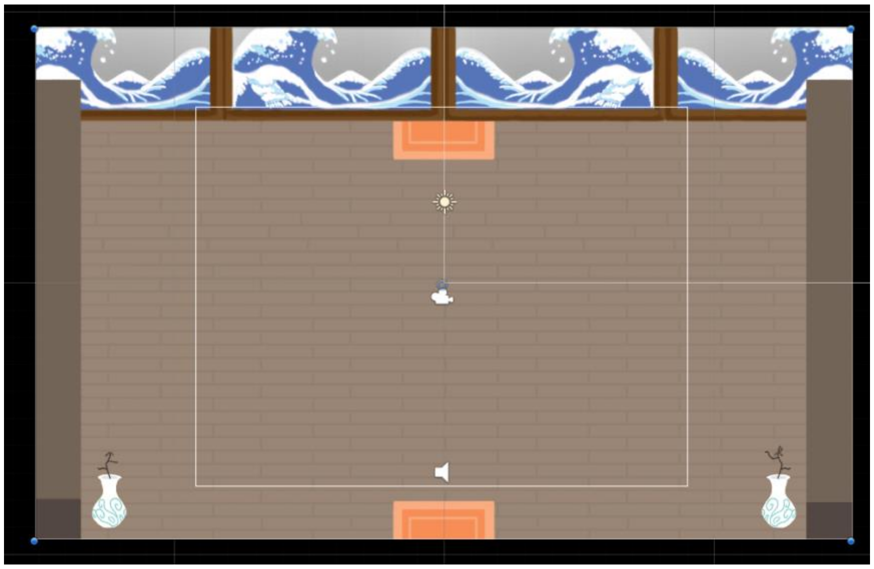


Figure 13: The background of our game

[1] Picture Taken at Entokuin Temple

3.3.3 Famous Painting “The Great Wave off Kanagawa”

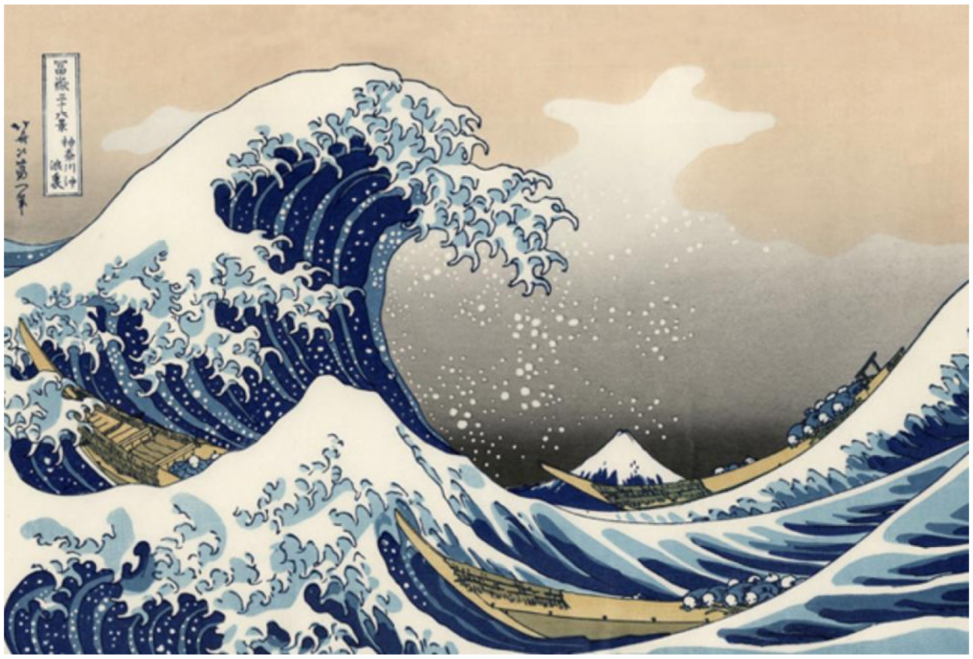


Figure 14: The famous Japanese painting “The Great Wave off Kanagawa” by Hokusai [49]

In the background, players would notice that there is the Japanese famous painting, The Great Wave off Kanagawa, painted on the door. The Great Wave off Kanawaga (神奈川冲浪裏 Kanagawa-oki nami ura, "Under a wave off Kanagawa"), also known as The Great wave or simply The Wave, is a woodblock print by the Japanese ukiyo-e artist Hokusai. It was published sometime between 1829 and 1833.[49]

We put it there to make it easier for the players, who don’t really know much about Japanese culture, to notice the Japanese-culture environment in this game, since this painting is a well-known painting from Japan and the style it paints make people think of eastern painting techniques.

3.4 Player Character Design

In the initial sketch of the main character, we wanted to have a character that is androgynous; a character that could be both view as female and male depending on the player perception. The reason that we wanted a character that could be of both gender because we wanted the player to be able to put themselves in the role of our main character as he/she plays our game. However, due to the team lack of an artist, when the complete sketch of the main character come, it looked more masculine than feminine. Changes needed to happen to get the right look for our character. However, around the time of our design, we felt that we needed to shift from our initial plan to create an RPG to only action game, due the time constraints. In addition, we felt that the character from the sketch did not fit the Japanese 2D game that we set out to make, so we needed to find a new style that fit that requirement.

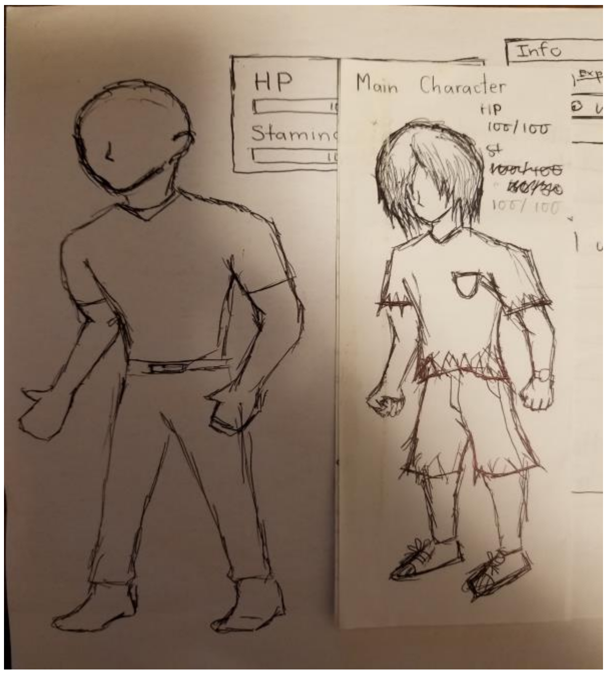


Figure 15: Kien's first sketch of main character

In Japan, video games that sold the most have designs that are colorful and have a more cartoonist style. Games like *Pokemon, Yokai Watch, and Legend of Zelda*. According to

Keiji Inafune, a video game producer, the reason for its popularity is that people in Japan grew up with anime and manga style of entertainment so they are more familiar with the style [6]. This can be seen on the streets, inside buses, advertisement, inside train station, and many other places in Japan. During our stay in Japan, we have seen advertisements about train created in anime style at Osaka Station and Kyoto Station. It is part of Japanese culture.

The game that inspired our character design is The Binding of Isaac and the Japanese subculture we discovered. The character design for *The Binding of Isaac* is fairly simple. The noticeable features are the large head and the large eyes. The approach to designing our character is drawing each body individually and eventually putting it together when animating it, making it easier to manage. Compared to the last design, this new design fit the style that we wanted to show in our game. It looked simpler and more manageable for a team without a true artist. In the initial sketch, the character has a bit of realism to it, but after discussion, having a realization that our game genre is fantasy makes it possible to have a character that is far from being realistic. In addition the style fits the current trend we found in Japan.

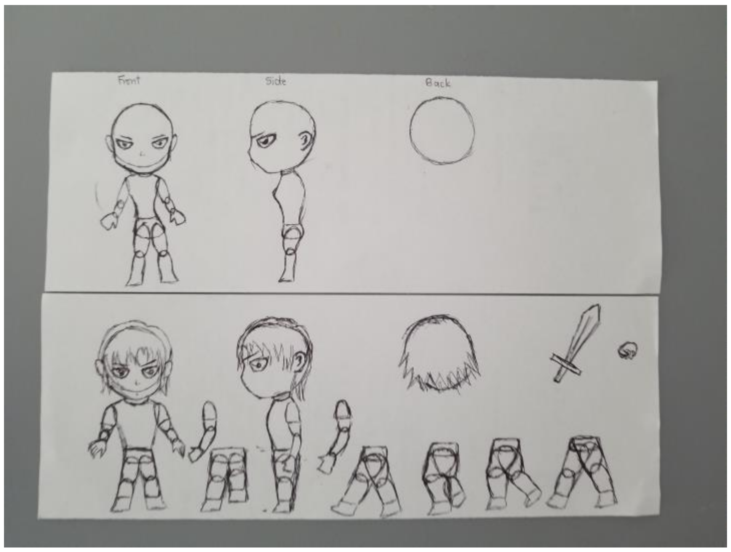


Figure 16: Kien’s second sketch of the main character

A widely known trend in Japan is what people refer to as, the “Kawaii” culture. Kawaii translate to cute or adorable in Japanese. This can go from clothes, accessories, hairstyles, and behavior to even physical appearance. In 2008, Hello Kitty, one of Japan oldest mascot became an ambassador for tourism for representing Japan [18]. Another mascot that we all know and love is *Pikachu* from the Pokémon series. There is even a store that dedicates itself to only sell Pokemon goods. The kawaii factor that exists in these two iconic characters is the disproportional bodies, big heads, wide eyes, a tiny nose, and little or no facial expression [18]. All of which can be seen in Kien design of the main character and enemy design. In Japanese arcade, there are pictures taking booth that allows people to have filters on their faces that is popular to having to make their eyes larger and rounder to achieve this “Kawaii” looks. While the connotation of the culture points the female population, it also extends to the male population as well. You have men trying to look younger by either having longer hair or shaving their legs [36]. The style is very distinct and it can be seen in Japanese Games like Pokemon, Yokai Watch, and Animal Crossings. These types of culture references can be found many JRPGs. According to Douglas, JRPGs are Japanese because its relevancy to the culture and trends in Japan [21]. We wanted to create a game that has a touch of Japanese to it and using the “Kawaii” sub-culture helps us make the game relevant.

3.5 Enemy Character Design

The Yokai that we used for the boss of our game is Nue. According to *The Tale of the Heike*, a Nue was viewed as a chimera. It had a face of a monkey, body of a raccoon dog, legs and arms of a tiger and a snake for a tail. There were other variations of its appearance so calling it a chimera is the most appropriate. Little is known about Nue, but in legends told for century, Nue were considered a “pretty evil monster” [30]. One widely known about Nue was that it makes a sound that is similar to a bird called White’s thrush [9]. Everytime the people hear the sound, people knew that a bad omen was approaching.

3.5.1 Visual Appearance

In designing the appearance of our game Nue, Kien, one of the artists of the team, used several references to design the Nue. The description of Nue found on Yokai.com by Matthew Meyer, is very straightforward and descriptive enough to start to begin the designing process. Following the same style as our character, Kien designed the Nue having a large head and small body. For reference, Kien used Nue from the more recent Persona game, Persona 5, for the color scheme and stylish look.

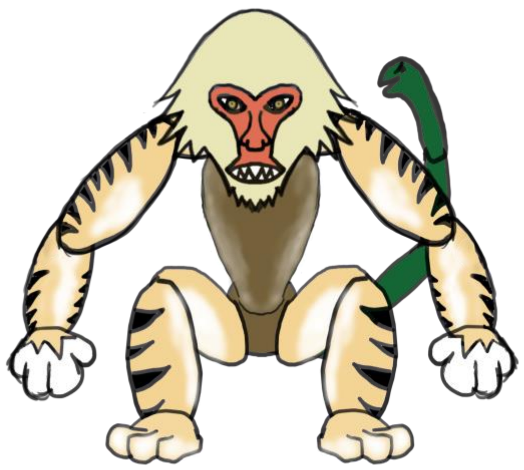


Figure 17: Kien final design of Nue

3.5.2 Personalities

The only common feature about Nue, other than its appearance, is the chaos and tragedy it is capable of causing. The detailed behavior and personality of Nue which brings destruction can vary greatly from different sources. Considering making them believable and matching the behavior Nue is known for, four distinct personalities are suggested and incorporated into the Nue agent.

Aggressive

Those possess the aggressive trait have common characteristics. They actively seek the superior position in any encounter. They do not want to submit to the rules around them. They are never comfortable with restrictions placed on them. They do not have sympathy for anyone. Finally, they have no mental brakes that prevent them from going too far [23]. Those with an aggressive personality want to emerge as the victor no matter what circumstances. Being the best drives them to behave destructively.

An individual with aggressive personality uses aggressive behavior as a means to relieve their “aggressive drive”. It lives by the reward and punishment system. If one gets a reward for an aggressive behavior, they continue to persist, while getting a punishment for such behavior causes the behavior to diminish over time. A factor that causes aggression in individuals is through social learning. One study, the Bobo doll study, showed that children learned how to be aggressive by watching others’ behaviors. As soon as children gain motor skills, they are more likely to display aggressive behavior because they are in their learning phase so they would mirror what they see. The age where they are more vulnerable to picking up habits is two years old. One typical example is that when a parent or an adult figure hit themselves, lightly, to make the child laugh, this reinforces the child mind that hurting oneself and other is funny. This means that aggression can be learned behavior. Another factor is genetics. Things like brain dysfunction, testosterone, and nutrition deficiency causes imbalance in the body, making the mind more susceptible to aggression than the average [24].

Aggressiveness is a primary trait of a monster or villain of any story. In the origin story told in the background of this section, Nue was a very evil monster. It repeatedly caused the emperor misfortune until it was killed. This inspired an idea that our Nue could possess a relentless trait showing its intelligence. Because Nue has an appearance of a wild animal or animals, it also possesses the aggressive nature found in any wild animals and even human beings. We decided that having an enemy that is wild and aggressive in its nature while having intelligence it was a great starting personality to model.

There are two categories of aggression, proactive aggression and reactive aggression [38]. Proactive aggression is aggressive behavior that aims to achieve some sort of goal. It is usually directed to a person or thing as a way to relieve the tension. Reactive aggression refers to an impulse or a reaction caused by anger. The difference between the two is that proactive is intentional while reactive is unintentional. In our game, we focus on modeling proactive aggression and the stimulus for its aggression is territorial and predatory. Because our boss character is a Yokai, it has characteristics of a human and animal. However we do not have a dialogue system in our game so we cannot model the character with complex emotion and personality. We can display the aspect of the aggressive personality through actions of the boss character.

Tricky

The only description of personality of this ancient Japanese monster is evil and people usually die when they encounter it. Therefore, we tried to interpret the evil of Nue in different ways and one of them is tricky, or a liar.

The reason we chose tricky as one of the personalities is, we want to create a monster who can compete with players on an intellectual level and win the game. The ones who win fights and battles are not always the strongest ones, but usually the ones with correct strategies and higher intelligence to trick their enemies [14]. In order to win a fight, monster needs to know when to attack and when to get away, which means it needs to keep a clear mind all the time. More importantly, the calmer a person is, the more likely this person is going to make the right decision [13]. Therefore, we pick tricky as a personality - it can always keep calm and stay to its strategy, and never be affected by its emotions.

This personality could be found on many representative villains in different games.
In the famous RPG Diablo, one of the villains named Belial is a demon that represents liar. He lies to both gods and devils to gain more power and to seize the throne of hell. [17] Characters with this kind of personality tend to be rational and choose the most efficient strategy to beat the player.

In our game, monster with this personality will try to trick the player with its movements and try to predict the movement of players in order to hit them. It takes in the moving direction of the player as variables and then when it starts to charge it adjusts its charging direction according to player’s moving direction. It will also try to get away from player while the charging mechanic of monster is still in cooldown. There is an immediate function for it to calculate the way to escape from player’s attack. A more specific introduction of this mechanic will be shown in its implementation section.

Bullying

Bullying usually involves using physical or verbal violence to dominate others [47]. Out of all subtypes, Aggressive bullies are the most common type of bully. Out of all subtypes, Aggressive bullies are the most common type of bully. They are aggressive and tend to dominate others physically. They see everything in the environment as hostiles, even though there isn’t any threat. There’s another kind of bully called Bully-victim, who have been seriously bullied themselves, and desperately trying to regain confidence and suppress negative feelings by bullying others who are weaker [27].

Mentioned several times in this section, Nue had always invoked fear in the average human. However, when Minamoto no Yorimasa injured it, it fell down and Ino Hayata finished it off easily [48]. It bullied people who couldn’t retaliate, but was instantly defeated by who was strong. Therefore we assumed that it could be a combination of aggressive bully and bully-victim, which means aggressive when it feels like it is stronger than the player, but scared and start to run away when it feels like the player is actually stronger.

Cowardly

Making a cowardly personality is taking everything suggested above and turns it to the exact opposite. What if Nue is very weak in its combat capability? According to a catchphrase from a movie *Akuryoto*, “Nights where the nue cry are dreadful” because of the disaster it brings, but it is never seen physically attacking or killing anyone [48]. Together with the fact that it was instantly finished off after exposed makes it possible to assume it was not good at fighting, and was only able to hide in the dark smoke and curse people. Because it loses all its witch power in our game, the only thing it can do is running away while trying to defend itself.

Chapter 4: Software and Tools

This section gives a brief introduction of the software, libraries or tools we used to develop our game, characters, animation and A.I.

4.1 Unity

Unity is a famous game engine which can output its products into different platforms [45].The reason we chose to build our game with this engine is that it’s a rather perfect and well-supported comparing to other engines. We didn’t choose unreal engine, since we are not making games with 3D models and it doesn’t have a complete developing environment for 2D games. Cocos2dx was another tempting option, however, like it’s mentioned above, we need a well-supported tool in order to implement our AI technologies and Cocos2d is obviously a worse choice than Unity which is maintained by hundreds of people and update the software almost once a month.

Another huge advantage we get by using Unity is that it supports almost every single platform that exists in the market. In 2018, the engine was updated to support 27 platforms, which means once we finish our game, we don’t need to make another version for mac and iOS or other platforms. [45] It would save us huge amount of time and effort so that we can focus on making our AI and the actual game.

Last but not least, Unity supports writing code with C Sharp, which is another reason we are building our game with this engine.

4.2 C Sharp

C# is a programming language. It was developed by Microsoft in 2000 and then approved as a standard by ISO and Ecma (ECMA-334). In addition, C# is initially designed for the Common Language Infrastructure, and this is why it is so widely used in famous software like unity [5].

We spent some time on getting familiar with the language and it definitely worth it. Unity combined its perks, such as visual game development, with C# code in an efficient way.

Since there are plenty of tutorial videos online and we all know Java – object oriented, we still learned it quickly although C# was not taught in any classes in WPI.

4.3 Spriter

Due to having a lack of an artist on the team, Kien became the primary artist for the character creation. Kien used software called spriter to animate the character walk cycle, attacking cycle, and idle cycle. Spriter is 2D animation software developed by BrashMonkey. It primary feature is animating through modular images or body parts. This is the reason that our character design is broken up into individual body parts. Using Spriter, modular images can be connected using the joint feature, which allow user to attach a “joint” to an image and allow users to connect individual joint together to create a fluid body movement. This feature allow easy tweaking because the modular images can easily be move or rotate to fit whatever position creators want to model. Spriter was perfect for our team because we needed to efficiently create character animations with less time spent of modifying as much as possible.

4.4 UtilityAI

UtilityAI is a utility based system framework developed by Bartvanderkruys. Initially it was used to help us further understand the implementation of a utility-based architecture. We decided to use it for our project when we realize the features that it offers was perfect for our project. It allows us to define several parameters called property to represent the status of the environment, and assign each action with utility functions based on certain properties. It automatically calculates utility score of each action by inputting current value of properties into utility functions, and calls the function linked to the action with the highest score.

Property values serve as status information of the Yokai or observations of the environment. Modifying how they change is one important way of modifying behavior, as they can increase or decrease at a faster or slower rate, controlling how utility score for each action changes for over time, thus changing the behavior (actions) of Yokai even if it goes through the same sequence of stimuli.

Utility functions define the relationship between property values and utility value (score) of an action, thus different actions can be invoked when different environments are present at different statuses. For example, by applying a positively related function of a property to an action, utility score increases with that property, meaning the action is more favored when that property value is higher. By applying a nonlinear function like y=x^2 (y is score, x is parameter), when compared to a linear function like y=x, utility score increases less when property value is less than 1, meaning less responsive to property change, but increases more when property value is more than 1, meaning more sensitive to property change. Tweaking utility functions is another important way of modifying behavior, as different utility functions means different scores and potentially different actions when the same set of properties are represent.

UtilityAI allows multiple functions of different properties for an action. They are calculated separately and scores for individual function are added together for the final score of that action. A modifier called weight can be added to each function, and the score for each individual function will be original score times modifier.

4.5 Panda BT

Panda BT[2] is a behavior tree framework from Unity Store. It is used by us to save time from making our own behavior tree system. We can create a tree by passing a built-in text file with script written in its own syntax, and invokes functions from c# scripts with a special [Task] flag. One of the features of Panda BT is the ability to observe the execution of the tree. There are three colors to the tree, red for failure, green for success and blue for running. Using this feature, it made debugging easier.

Panda BT has 10 structural nodes but we only familiarize ourselves with only 6 of them in our BT script. They are tree, sequence, fallback, parallel, while, and not. Tree is used to create a tree, while the others decide how the node in the tree should execute. Sequence node run its children in sequential order and only succeeds when all the children succeed. Fallback node runs its children the same way as sequence node except the node only succeed when one of its children succeed. Parallel node runs its children all at the same time and only succeed when all the children succeed. A while node has a condition and an action. It succeeds when both succeed. The not node is used in the same logic in any programing language; it succeed if the condition is true, fails otherwise. The mute node always succeeds. These structural nodes make up Panda BT.

[2] Documentation at http://www.pandabehaviour.com/?page_id=23

Chapter 5: Implementation

5.1 Intro

In this section we are going to talk about how we implemented each part of this game, including game mechanics and each AI personality. The description of implementation will be specific and precise.

5.2 Class Design

Our game, Yokai World, is programmed by C# in unity. For most of the games made with unity engine, including ours, there are no main classes in their code.

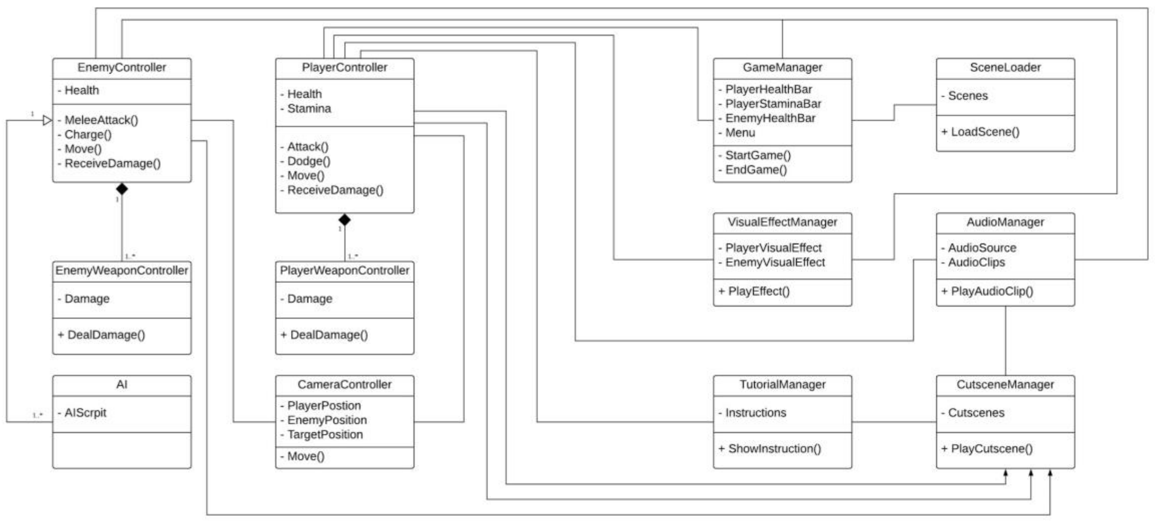


Figure 18: Class Design Diagram

In the figure “Class Design Diagram”, there are two main controllers, which are enemy controller and player controller. The whole game is designed based on these two controllers, because the actions of these two controllers will be the main factors of this game. Each enemy weapon controller is attached to an enemy, and each player weapon controller is attached to a player. One player can keep multiple weapons but one weapon can only be attached to only one player.

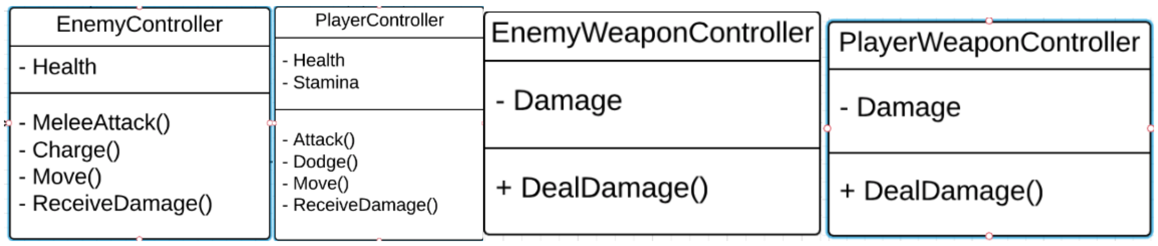


Figure 19: Player and Enemy Controller

Figure 19 shows attributes and methods in enemy, player and their weapon controllers. In the enemy controller, there's only one attribute, which is health. In player controller there are two attributes, which are health and stamina. There are also different methods in these two controllers which stand for their different abilities and have been introduced in game design.

However, note that there is a method called receives damage in both of the controllers. To introduce this method, we need to talk about the weapon controllers first. The two weapon controllers have an attribute called damage and this determines how much damage they can deal to the opponent. After weapon controller collide with an opponent, the deal damage method will call the receive damage method in opponent's controller and deal a certain amount of damage to the opponent.

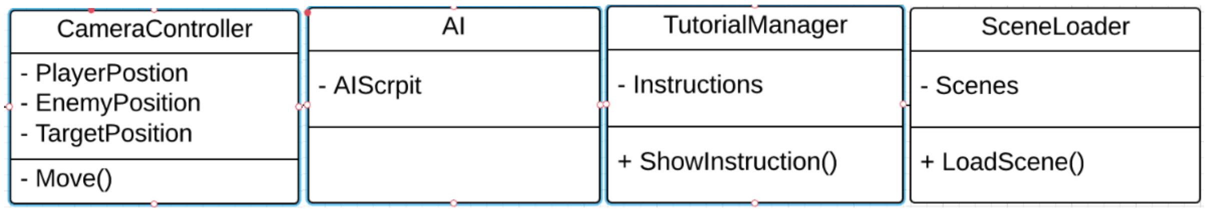


Figure 20: Camera AI Tutorial Scene Controller

Camera controller, as shown in Figure 20, takes in the player's position and enemy's position and calculates the correct position for the camera to keep both player and enemy in the shot.

There’s also an AI attached to each of the enemy controllers. Each AI contains a unique AI script which let them operates with different personalities. AI scripts can be a behavior tree, a state machine or behavior tree based on utility.

The tutorial manager contains interactive instructions for the player, which means players can learn the control of game by going through a small tutorial session. The instructions will be played when the show instruction methods is called.

Scene loader contains the name of different scenes. When a scene transfer is needed, load scene method can be called to load another scene.

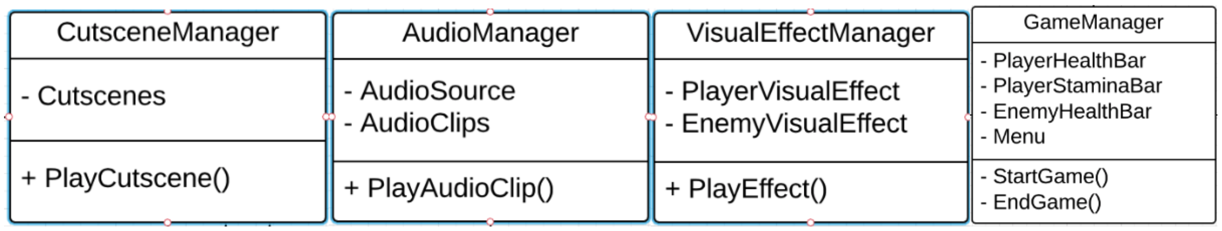


Figure 21: CutScene, Audio Visual and Game Manager

In Figure 21, it shows that cut scene manager contains several cut scenes. Its mechanic is just like the mechanic of scene loader. It plays a cut scene when the play cutscene method is called.

Audio manager takes in audio clips and play the clip by an audio source. The clips will only be played when the play clip method is called.

The way that visual effect manager works is similar to the mechanic of audio manager. It takes in animations of visual effects and then they play it with an animator attached to the object. The effects will only be played when the method play effect is called.

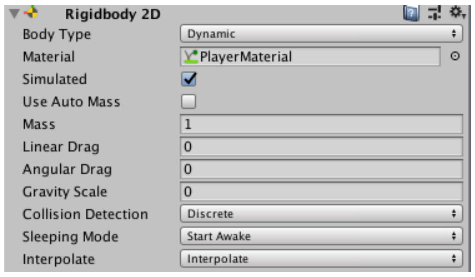
In game manager, it operates all the user interface elements, including health bars and menu. The most important methods it contains is starting the game and ending it.

5.3 Technical Implementations in Unity

5.3.1 Player & Enemy & Camera

Fundamental objects in Unity that represent characters, props and scenery are called gameobject, including player and enemy. Player and enemy controller are attached to the player and enemy gameobject. They share the same mechanic of movement, except that the function for movement in player controller takes input from keyboard and the function in enemy controller is operated by its AI.

In player and enemy controllers, there is an element called "rigidbody" and its velocity is changed by a vector. Rigidbodies are attached to gameobjects and enable them to act under the control of physics. There are different attributes in the rigidbody, including its material, mass, gravity scale a body type. These attributes will determine its physical features under the physics engine. Therefore, adding rigidbody to gameobjects and modifying its velocity can make them move in a more realistic way.



Rigidbody 2D	
Body Type	Dynamic
Material	PlayerMaterial
Simulated	✓
Use Auto Mass	☐
Mass	1
Linear Drag	0
Angular Drag	0
Gravity Scale	0
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	Interpolate

Figure 22: Rigidbody component in Unity

Besides simply moving by changing the velocity of its rigidbody, player has another choice of moving.

```
private void Dodge(){
    if player press "Space" and player is moving up{
        player.position.y = player.position.y + dodgeDistance;
    }
    ...
}
```

Code 1: Pseudocode for dodging

The dodge function let player dash through a distance in one selected direction. It takes in keyboard input "Space" and a direction key ("W, A, S, D") to operate.

```
private void attack(){
    if player press "J"{
        player.animator.layerplayAttackAnimation();
    }
}
```

Code 2: Pseudocode for attacking

The main effect of melee attack function of players and enemies is to play their animation of attacking. When the animation is played, weapons will move and they collide with opponent, they will deal call the deal damage function to hurt its opponent.

```
private void DealDamage(Gameobject enemy){
    enemy.ReceiveDamage(damage);
}

private void ReceiveDamage(int damage){
    health = health - damage;
}
```

Code 3: Pseudocode for dealing and receiving damage

The special attack of monster, charging, is just simply locate the player’s location and charge. Different personalities will have their own way to determine the players' location and ability cooldown.

```
private void LocateCamera(Gameobject player, Gameobject enemy){
    if player and enemy are not null {
        camera.position = (player.position + enemy.position)/2
    }
}
```

Code 4: Pseudocode for moving Camera

Camera is one of the most important gameobject in unity. It offers all the visual feedback to players. Normally camera would be following the player. In our game, however, player needs to know the location of monster as well as their own position. Therefore, we created a method to take in both player's position and enemy position to calculate a suitable position for camera. In the pseudo code above, there is an illustration of the method.

5.3.2 Animation

Animation is one of the most important part in an action game, since players need visual effective feedback to fight with their enemy. In our game, animation is implemented by two built-in mechanics in unity - animator and animation.

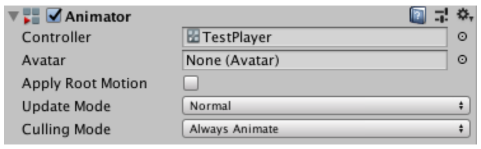

Animator
Controller: TestPlayer
Avatar: None (Avatar)
Apply Root Motion: ☐
Update Mode: Normal
Culling Mode: Always Animate

Figure 23: Animator Component in Unity

As shown in Figure 23, animator can be attached to a gameobject in the scene, and it requires a reference to an animator controller which defines which animation clip to use and controls the transitions between them.

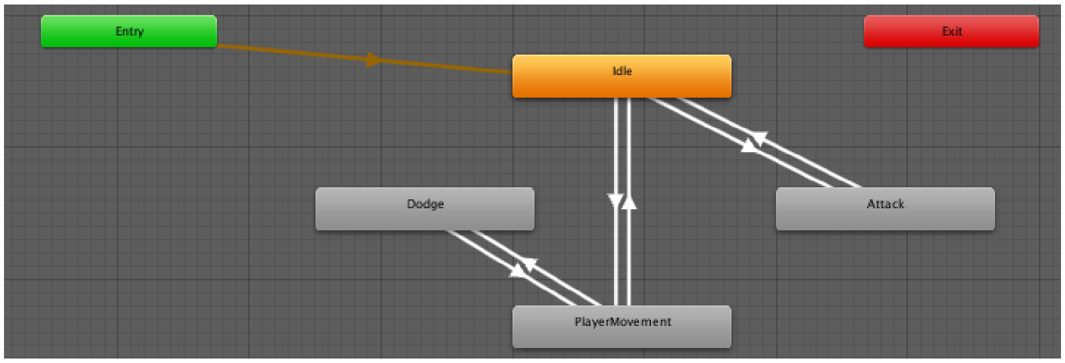
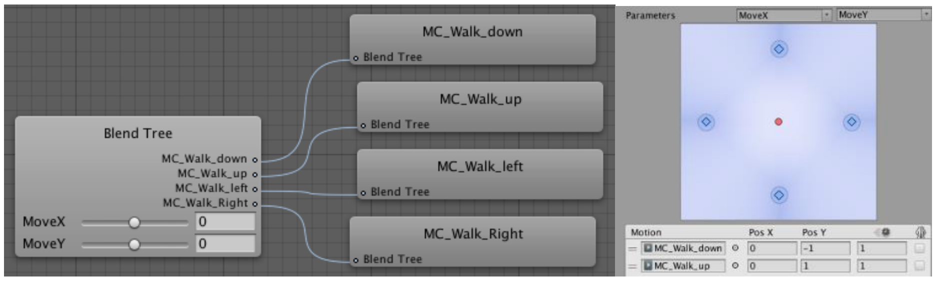


Figure 24: Animator Controller

Figure 24 is an example of animator controller in our game. Each block stands for either a clip of animation of a blend tree. The transitions between them are represented by the lines arrows which linked them together. They will transfer from one clip to another when a certain condition is met, which can be a Boolean, a trigger or a float.

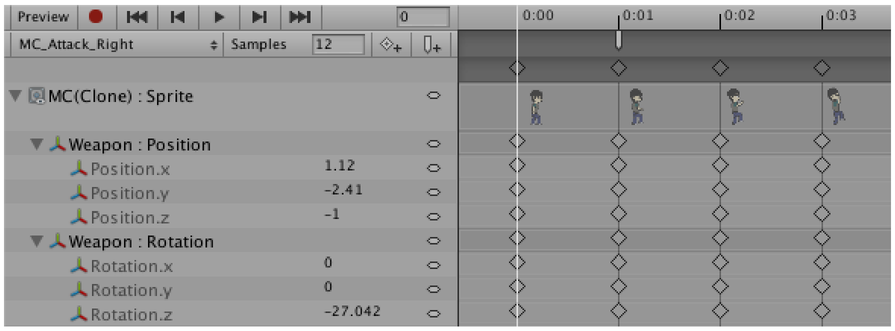


MoveX 0
MoveY 0

Motion	Pos X	Pos Y	
MC_Walk_down	0	-1	1
MC_Walk_up	0	1	1

Figure 25: Blend Tree

Blend tree, as shown Figure 25, is different from normal animation clips. It can take in multiple values and visualize the manipulation on different clips according to these values. It's a blend tree for walking. The red point in the middle of the right figure stands for the current direction of walking. When the player is walking up, the point goes to the top, when player walks right it will move to the right side. With this mechanic, player is able to move diagonally with an animation, since the red point can move diagonally according to the input values.



MC(Clone) : Sprite
Weapon : Position
Position.x 1.12
Position.y -2.41
Position.z -1
Weapon : Rotation
Rotation.x 0
Rotation.y 0
Rotation.z -27.042

Figure 26: MC Attack Animation

With the other mechanic in unity, animation, we are able to adjust each animation clips frame by frame. In Figure 26, on the left side is the manual where we select clips and attributes of a gameobject to edit. On the right side there is the panel for frame by frame edition. Each rhombic piece on each frame stands for an attribute. The small narrow bar right beneath the time bar stands for a function that is triggered on that frame. Most of the interactions between gameobject that involve animations are done by this mechanic.

5.3.3 Collision

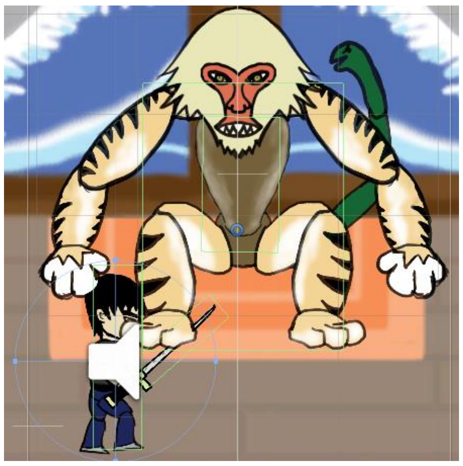


Figure 27: Box Colliders

As shown in Figure 27, there is a green box attached to the enemy, player and their weapons. These green boxes are called box colliders. Box collider is a cube-shaped collision detector. Each of them is attached to one gameobject, and this is the unique way of unity to detect collisions. If the box collider is set up as a trigger, it can trigger different functions when it collides with other box colliders and it will not block others' way. If it's not set up as a trigger, however, it will block other box colliders.

In Figure 27, the box colliders on weapons are triggers so they can collide on other colliders and overlap on other game objects. The player and monster got non-trigger box colliders so that they cannot overlap on each other.

5.3.4 Audio Effect

Sound implementation is done by several steps. First, we found all the free music and sound effects resources that we need in our game. Most of our sound effects are from a website called Freesound.org.

After downloading the clips from the websites, we did some simple modification to them, such as cutting out the parts which we don't need, and splitting one clip into multiple clips. Note that all of these clips are in .wav format so that it's easier for us to modify them in the future if needed. All these modifications are done by Audacity[3]. Then we import those clips into a folder called "audio" in unity.

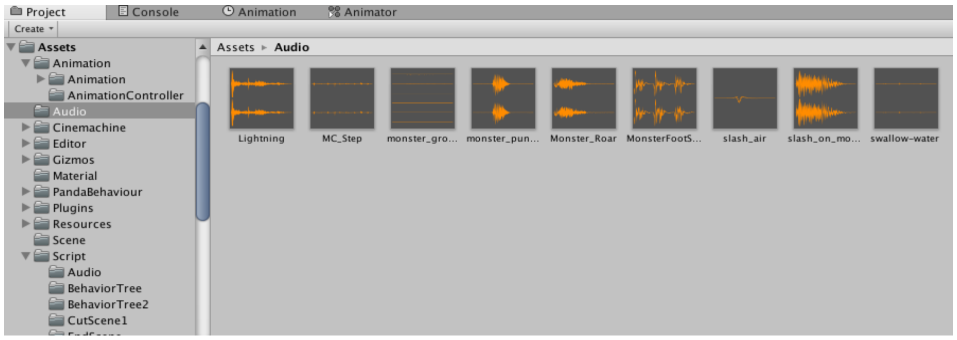


Figure 28: Audio clips imported into unity

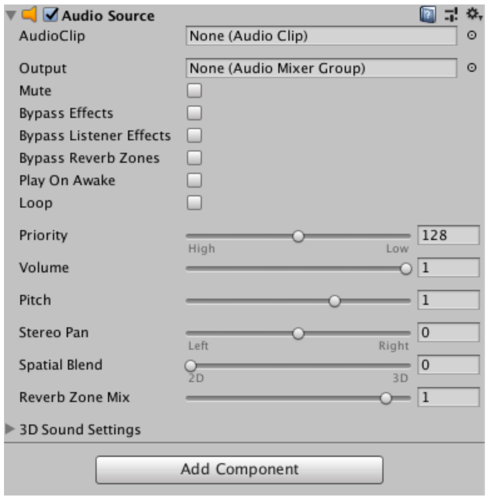
To use these clips, we created a script for them, which are called Audio Manager. Audio Manager contains methods that can be called in other classes to play these clips, and it also take in all the clips as variables. For example, in the figure below, "slashAir" is an audio clip and "PlayPreChargeSound" is a function which can be called in other classes to play a clip.

```
public AudioClip slashAir;
public AudioClip MonsterFootStep;
public float volume;

private AudioSource source;


// Use this for initialization
void Start () {
    source = GetComponent<AudioSource>();
}

public void PlayPreChargeSound(){
    timeToGrowl++;
    if (timeToGrowl < 1)
    {
        source.PlayOneShot(monsterGrowl, volume);
    }
}
```

Code 5: Code for Playing Charge sound

[3] Audacity is a free and open-source digital audio editing software.

For players to hear the sounds, however, we still need to add an audio listener and several audio sources to the game objects in the scene. Audio listener is a component of a game object which can detect sound. It can get input from audio sources in the scene and then plays sounds through the computer speakers. In this game, we attached the listeners to the camera and camera will stick to players and enemy so that all the sounds made by player and enemy will be played.



Audio Source

AudioClip	None (Audio Clip)
Output	None (Audio Mixer Group)
Mute	☐
Bypass Effects	☐
Bypass Listener Effects	☐
Bypass Reverb Zones	☐
Play On Awake	☐
Loop	☐
Priority (High – Low)	128
Volume	1
Pitch	1
Stereo Pan (Left – Right)	0
Spatial Blend (2D – 3D)	0
Reverb Zone Mix	1

Figure 29: Audio Source to Play Audio in game

After all these steps are done, players should be able to hear those clips played while playing through the game.

5.3.5 Visual Effect

Child game objects were created and attached to monster and player. Each of these child gameobject on player and enemy contains an animator component and several animation clips. These clips are the special visual effects that will be played when monster or player gets hit. As it is shown in Figure 30, for example, there is a red slash on the monster. It is actually the animation of the monster when it gets hit by player. It’s consisted of two frames which means it happens rapidly and creates feeling of a real slash by sword. This animation can be played in other directions randomly in order to create a variety of slash. These simple mechanics is generated by code in C#.

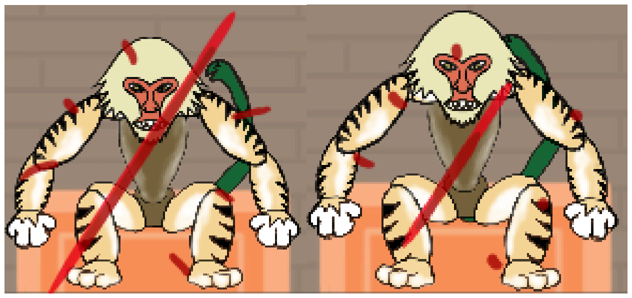


Figure 30: Slash Effect on Nue

There is also another effect, camera shake, in this game. When player hits the enemy with his sword, the camera will shake in a small range to add a sense of motion into players attacking action. This effect was not added to the enemy’s punch on player because we thought it would be a distraction for player if camera shakes every time they got hit by the monster.

5.3.6 Tutorial

Tutorial session is an important part of this game, since it is the only way for players to learn to play this game. We wanted to create an interactive tutorial instead of a page showing the control of this game.

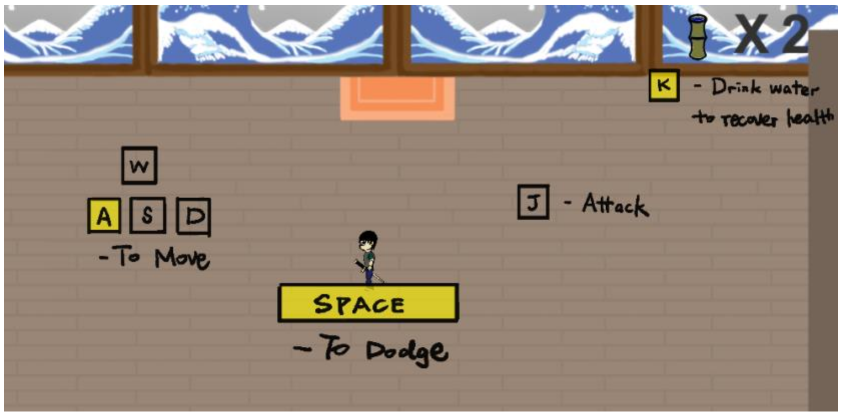


Figure 31: Tutorial for Our Game

As we can see in Figure 31, all the keys to cast abilities shows up before the monster enter the scene. The reason that these instructions will shine is to notify players to press certain keys and the keys been pressed will turn yellow in the tutorial which indicates that the player has pressed the right key. These instructions will disappear after players complete them and players can go to the next scene only when they complete all the instructions.

5.3.7 Cutscene

We wanted to add in a cutscene in our game because we wanted to make a smooth transition from one game scene to another. When the game starts, we wanted to give player time to get drawn into the game using a cutscene of the player waking up in a room as soon as the game starts. Because the player must fight the Yokai multiple times, we created another cutscene to show how the monster survive and why the Yokai appears in the next scene. It helps ties the game together.

The cutscenes are done using a combination of importing animation to Unity and through C# code. The animation where the main character wakes up in the beginning is done using spriter to animate the movement. The part where the enemy, Nue moves toward the player in the first cutscene and the scene where the Nue runs away from the player is done through a coroutine function that runs in parallel with other functions. Using the yield statement inside the coroutine, we were able to allow the function to run continuously which time the actions nicely to execute a simple an in-game cutscene.

The pseudocode in Code 6 prevents the moving while the waking up animation for 2 seconds. Then when the player finally is able to move on to the next scene, keyClicked returns true which starts the next portion of the surprise() function. The portion of the function is the animation of Nue pushing the player to the center of the scene and walking toward it. The scene is eventually changed to the battle scene after the cutscene ends.

```
surprise(){
 yield return new WaitForSeconds(2.0f);
 playerCanMove is true;

 yield return new WaitUntil(keyClicked);
 enemyAppear is true;
 PushPlayer();
 SceneFadeIn();
 PlayLightningSound();

 yield return new WaitForSeconds(1.0f);
 enemyIsMoving is true;

 yield return new WaitForSeconds(2.0f);
 StartNewScene();
}
```

Code 6: Pseudo code for a Cutscene

5.4 Personality

5.4.1 Aggressive

Implementation Structure

For the aggressive personality, Kien used a combination of Behavior Tree and Utility using two frameworks to assists us, Panda BT and UtilityAI. Kien felt very familiar with behavior tree and because utility-based architecture has the ability to give the AI autonomy, he felt it was the best way to create a semi-autonomous AI.

Figure 32 shows a selector node, “How Should I Move”, that decides which of the three sequence node to execute, Charge, Walk, and Run Away, using a utility function. This selector tree encapsulates the three different movements that Nue can choose at any given time in the game. The tree can execute any action sequence at random or in order; none of the behaviors are wrong choices to make, but with the utility function, it will select the best.

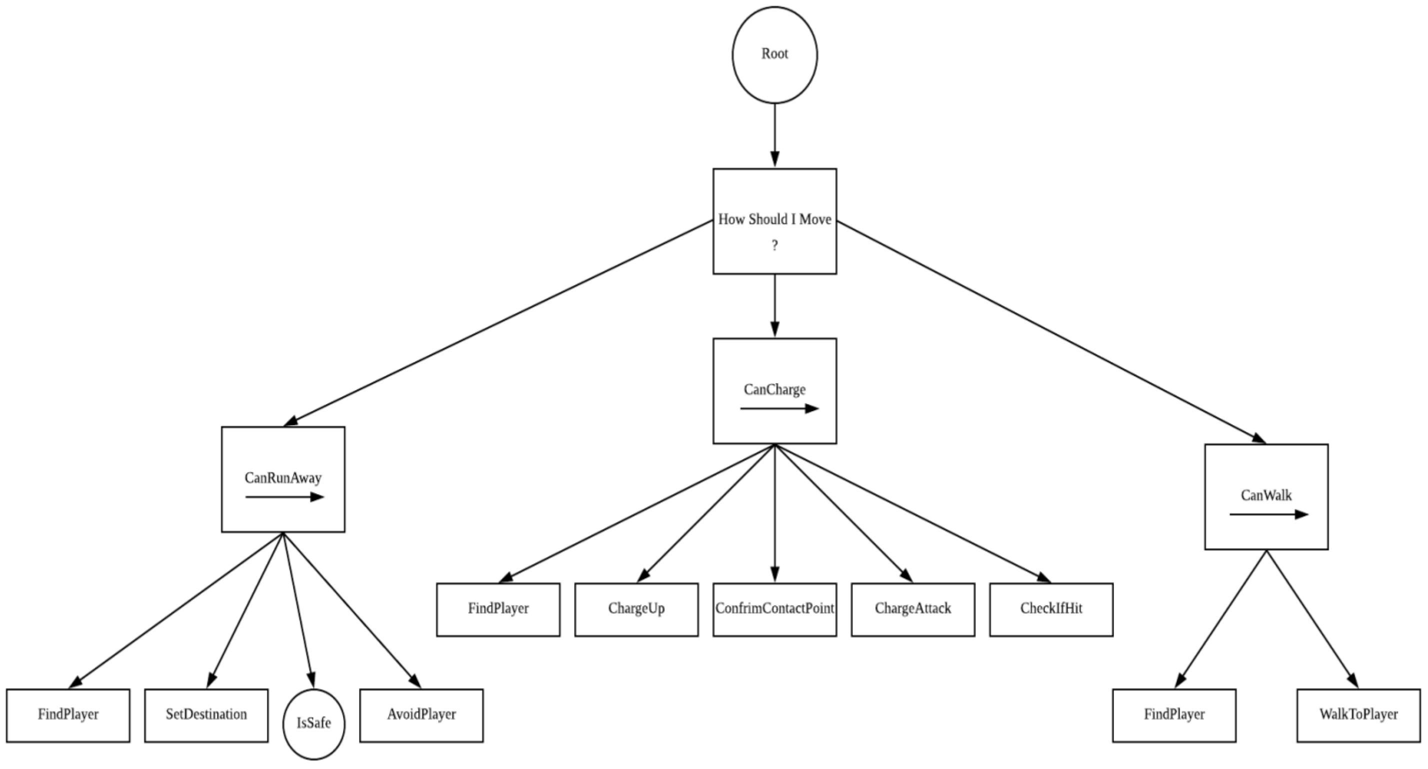


Figure 32: Aggressive Behavior Tree with Utility as Selector

Figure 33 shows the RunAway sequence node of action for the AI to perform. It first finds the player. Then set a destination on the map to reach. Then it checks whether the location is safe to walk to and if yes, the AI will proceed to walk to the location.

The SetDestination function randomly selects a position on the map and set that position as the destination to arrive at. The IsSafe function is a Boolean function that checks if the direction of the final destination is close to the player by a radius of 3.0 float and return true if the direction is safe. If the function returns false, then the action is reset and reevaluate the utility scores of the three actions.

```
public bool SetDestination(){
 destination = Random_Position_On_Map;
 return true;
}
```

Code 7: Pseudocode for setting a destination

```
public bool IsSafe(){
 if distance from destination to player in < 3
  return true;
 else
  return false;
}
```

Code 8: Pseudocode for checking if destination is safe

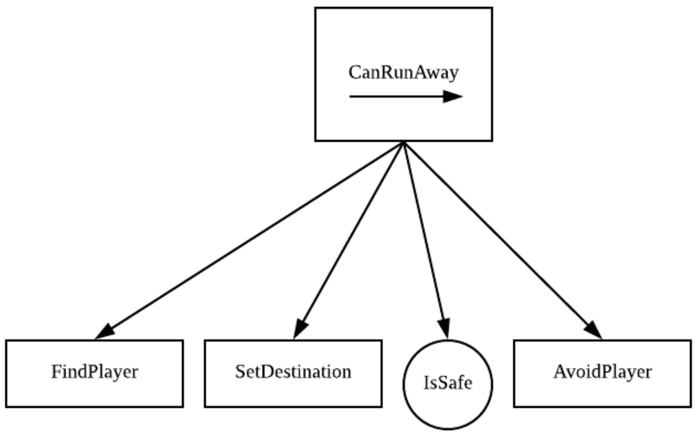


Figure 33: Run Away Sequence Node

Figure 34 shows the Walk sequence of action for the AI to perform. It first finds the player and then the AI will walk to the player after figuring out the direction of the player.

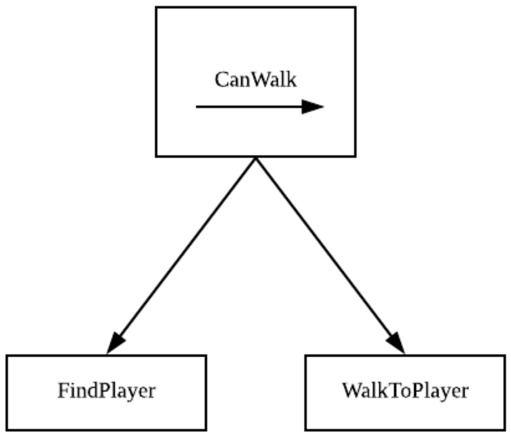


Figure 34: Walk Sequence Node

Figure 35 shows the node sequence for charging action. It first looks around the environment for the player. Once complete, it will then charge up its power and then it will reconsider its decision before charging at the player with the confirm contact action and finally it then charge at the player. ConfirmContact() function checks whether the player last recorded position and player current position is the same and if not, the new direction will become the final direction before savagely charge to that direction.

The last action is the reaction that the AI agent received from whether its attack landed or not. Depending on hit or miss, it changes the agent emotional state, which affects its action later on. The function CheckIfHit() function will always execute because the ChargeAttack() function will always succeed. What the function primarily do is modifying the property's value, mentioned later this section, according to whether the Nue landed the attack or not. Missing an attack results in becoming angrier, lowering calmness state, slightly increasing anxiety and losing energy from energy exerted performing the action.

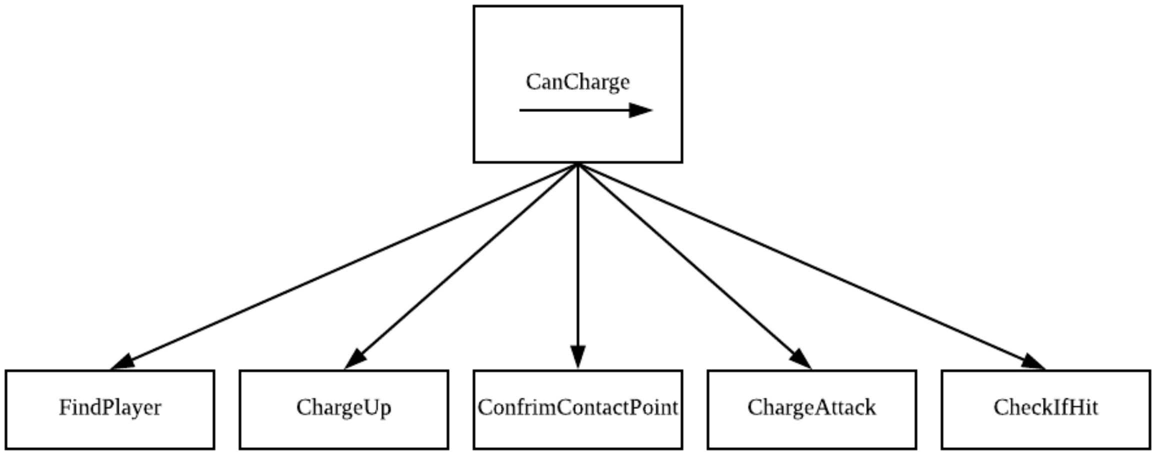


Figure 35: Charge Sequence Node

```
public bool ConfirmContact(){
 if distance from destination to player is not 0.5 then
  change destination to close to player;

 return true;
}
```

Code 9: Pseudo Code for ConfirmContact

```
public bool CheckIfHit(){
 if nue attack missed then
```

```
  energy value goes down;
  anger value goes up;
  calmness value goes up ;
  nervous values goes up;
  nue_miss_counter++;
 else
   energy value goes down;
   calmness value goes up;
   nervous value goes down;
   nue_hit_counter++;
 return true;
}
```

Code 10: Pseudo Code for CheckIfhit

Figure 36 is the Combat sequence tree. It has only one behavior, Melee. This Melee action is not part of the utility calculation because it makes more sense for an aggressive character to attack any chance it can get, regardless of the utility score. The Melee action has a sequence of conditions that must be fulfilled before executing its attack. Using Panda BT as a framework, the Combat tree is checked congruently with its siblings. By doing so, it will always sense the player at any time and attack when an opportunity come. Adding Melee to a list of actions for consideration can potentially make it appear less aggressive since the utility score for Melee may always be lower than the others and could cause it to attack less often than the design want it to.

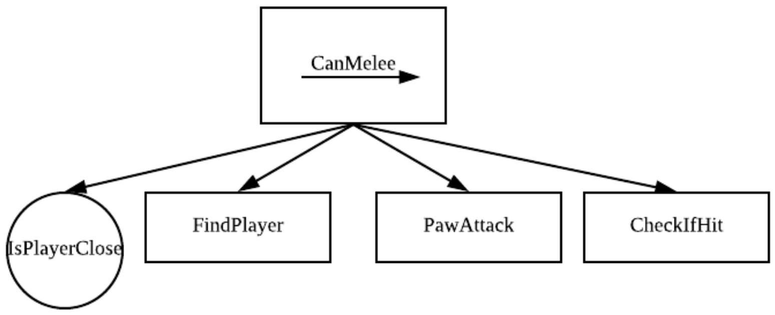


Figure 36: Combat Sequence Node

Figure 37 shows the observation selector node that the agent must execute. There are no true actions in this node as it is used for the agent to sense the player's movement and action at each update (one second per frame). This node is primarily used to model the aggressive personality by simply adjusting the properties, explained in the next section. Because the aggressor is easily angered and annoyed by the action of objects, living and non-living, around it, this node shows the different ways in which the AI can get agitated. For instance, the more the player goes on the offensive by landing its attack on the agent, the angrier the agent gets. The ModifyState action is simple function that modifies the value of each property by increasing it by some factor.

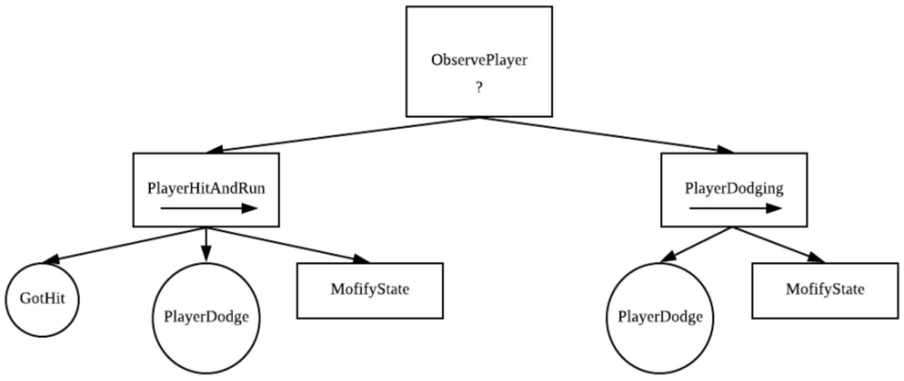
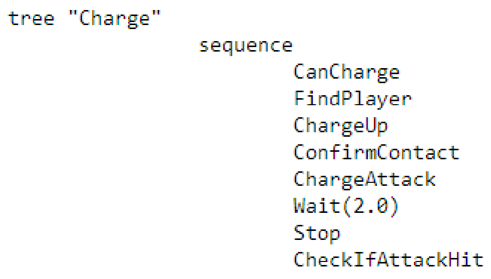


Figure 37: Observe Selector Node

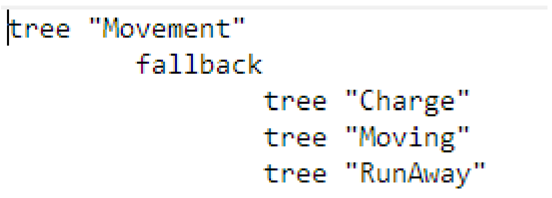
Panda BT and UtilityAI

We used the Panda BT framework for creating the behavior tree. Code 11 shows an example implementation of creating a Charge sequence tree. It is done by using a BT script that came with Panda BT. The root “Charge” indicates the starting point and everything under it is the actions to execute. By declaring sequence, it indicates that the behaviors below will be done in from top to bottom and only succeed if all the actions succeed. When the sequence node succeeds, it will return to the parent node as seen in Code 12. In this figure, there are three trees “Charge”, “Move”, and “Run Away”, similar to Figure 35 minus the details of each branch. All three tree are executed using fallback node, in another term, selector node, and the tree succeed when one child succeed and fails when all children fails. This is generally how we created a tree in Panda BT.



```
tree "Charge"
                sequence
                        CanCharge
                        FindPlayer
                        ChargeUp
                        ConfirmContact
                        ChargeAttack
                        Wait(2.0)
                        Stop
                        CheckIfAttackHit
```

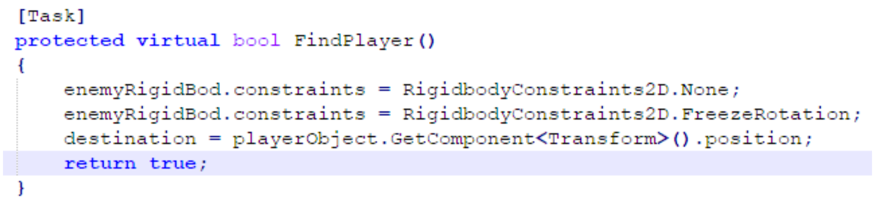
Code 11: Charge Sequence tree in Panda BT script



```
tree "Movement"
        fallback
                tree "Charge"
                tree "Moving"
                tree "RunAway"
```

Code 12: Selector tree Panda BT script

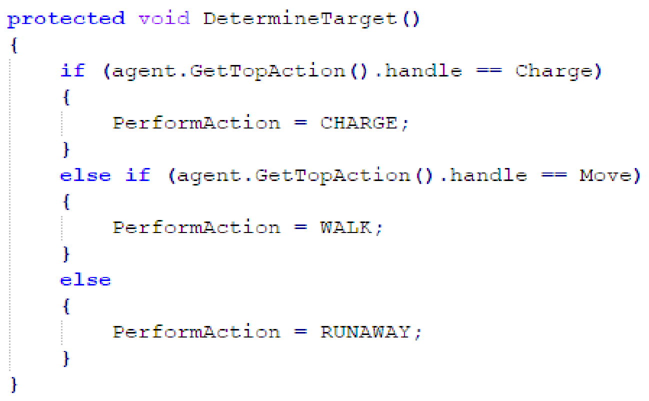
Each function that defined in the BT must use the tag [Task] attribute in C# code. Nothing changes from their functionality other than the fact that the function can be executed together with Panda BT script.



```
[Task]
protected virtual bool FindPlayer()
{
    enemyRigidBod.constraints = RigidbodyConstraints2D.None;
    enemyRigidBod.constraints = RigidbodyConstraints2D.FreezeRotation;
    destination = playerObject.GetComponent<Transform>().position;
    return true;
}
```

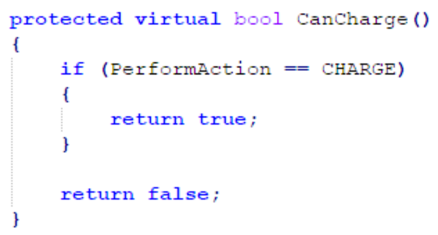
Code 13: A function declared as a task for BT script

Panda BT is combined with the functionality of UtilityAI by executing the calculation using its own update function UpdateAI() in the Update() in the C# script attached to Nue. We used an int variable named PerformAction to determine the action to take based on the UtilityAI decision from the calculations. PerformAction will always be set to 0 to disallow any sequence from activating. Any other number would represent different action to take, for examples, Charge is 1, Move is 2 and RunAway is 3. In Code 11, there is a task called CanCharge which would only return if and only if PerformAction is 1 (Charge = 1), if not the node will fail preventing any further execution. DetermineTarget() gets the top action of the agent and PerformAction will be set to a numerical equivalent of the top action. Panda BT and UtilityAI runs separately but we made the two work together using the two functions in Code 14 and Code 15.



```
protected void DetermineTarget()
{
    if (agent.GetTopAction().handle == Charge)
    {
        PerformAction = CHARGE;
    }
    else if (agent.GetTopAction().handle == Move)
    {
        PerformAction = WALK;
    }
    else
    {
        PerformAction = RUNAWAY;
    }
}
```

Code 14: The function determines which action to take after calculation



```
protected virtual bool CanCharge()
{
    if (PerformAction == CHARGE)
    {
        return true;
    }

    return false;
}
```

Code 15: The function that determines whether to execute the tree nor not

Properties

In modeling an aggressive personality, an agent needs properties or state that the utility functions can evaluate. All of these are made possible with UtilityAI UI. The states that we ended up with are anger, calmness, nervousness, and energy. Using the UtilityAI, we could define the maximum and the minimum of each property. Because the personality is aggressive, the anger state is set with a higher minimum than calmness and nervousness because an aggressive being is more susceptible to anger.

Actions

The Nue agent must have an action to evaluate so there are three actions that the utility function needs to decide on. The actions are Charge, Move, and Run Away. Each individual action needs considerations in order to make a decision. Charge takes into consideration of anger and energy.

Charge action is given the highest priority of the other because it represents the most aggressive behavior. It deals the most damage while also making it vulnerable. As a boss character with an aggressive personality, it should care more about the outcome rather than the process. Meaning dealing more damage is more important to the Nue in gaining a dominant position in the battle.

Figure 38 shows an example of Charge actions and its consideration. Anger and energy has the same weight of 1 while nervous has a lower weight of 0.6. The nervous in this instance represents the irrationality of the agent. This shows that anger is not only a factor to make the agent aggressive, but being in a high nervous state can cause the agent to interpret the situation as dangerous and thus enter fight or flight situation; in its case, fight. The utility score of this action is the average of the three considerations.

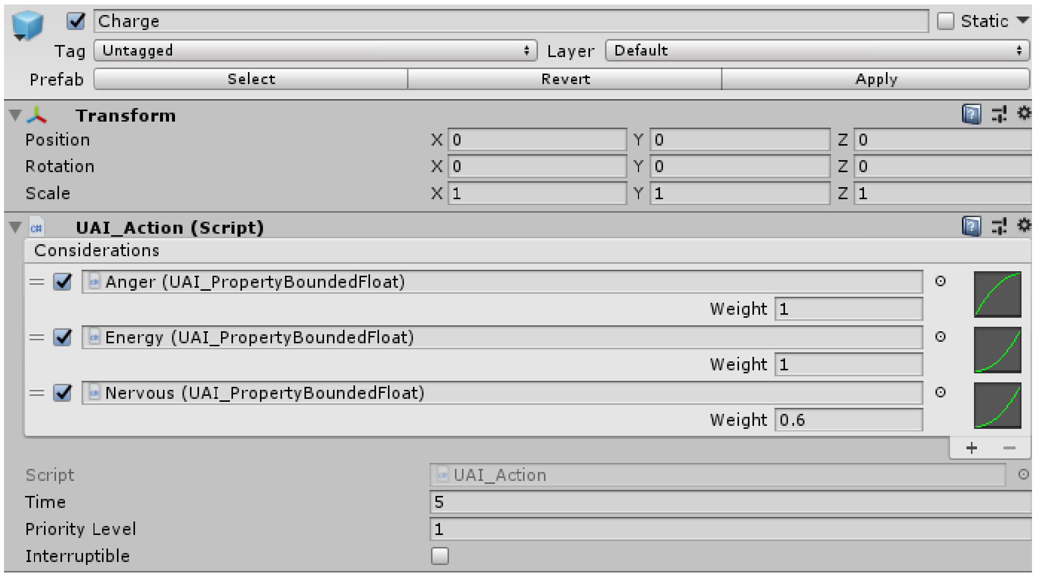

```
[✓] Charge                                   [ ] Static
Tag Untagged                 Layer Default
Prefab  Select   Revert   Apply
Transform
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale      X 1   Y 1   Z 1
UAI_Action (Script)
Considerations
[✓] Anger (UAI_PropertyBoundedFloat)
                     Weight 1
[✓] Energy (UAI_PropertyBoundedFloat)
                     Weight 1
[✓] Nervous (UAI_PropertyBoundedFloat)
                     Weight 0.6
Script          UAI_Action
Time            5
Priority Level  1
Interruptible   [ ]
```

Figure 38: UtilityAI UI Charge Action with Considerations

The anger value and energy fluctuates frequently throughout the battle. Nue energy lowers when it charges, attack, and getting attacked. The reason to have energy as a consideration because we want to make the Nue be more realistic in its action. It would not make much sense if Nue charges all the time without much constraint. While it does make sense for a monstrous beast to have unlimited energy and deals big damages all the time, it would make Nue to be an unrealistic one-dimensional character.

Anger consideration uses a logarithmic function. The function rises quickly from utility score of 0 and eventually level off near the utility score of 1. The anger value has the most impact in the beginning of the graph. Energy consideration uses a exponential function. It starts off slowly and rises exponentially. Because energy is important in performing a high demand action like charging, it requires high amount of energy to maintain it. Therefore, when energy is low, the utility score would also be low to prevent the agent from charging. Nervous consideration also uses an exponential function.

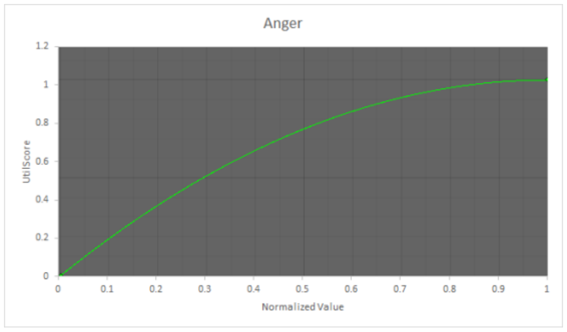


Figure 39: Anger Function for Charge Action

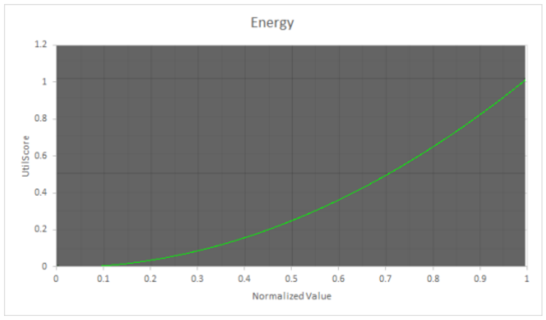


Figure 40: Energy Function for Charge Action

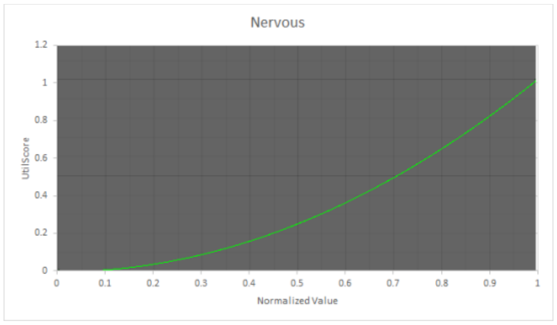


Figure 41: Nervous Function for Charge Action

The Move action takes into calmness and energy properties into consideration when choosing to move normally. Calmness consideration uses a sigmoid function with a weight of 0.6. The function rises the quickest in the middle of the curve. Having a moderate amount of calmness made the agent to take on a more calming behavior. Energy consideration uses an inverted logistic function. Energy is important and when energy is becoming low, it is important for the agent to make use of the move action to act logically by focusing on recover energy rather than act rashly by charging more. This happens when energy is close to the half mark and using the opportunity to conserve as much energy as possible would make it appear intelligent.

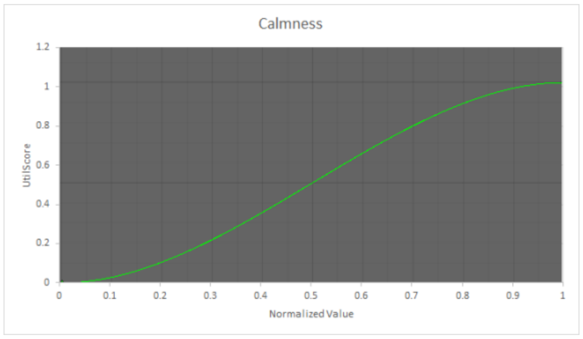


Figure 42: Calmness Function for Move Action

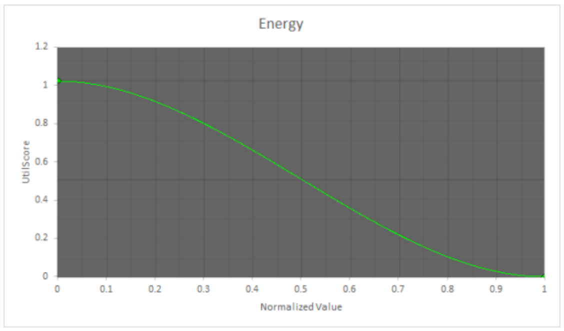


Figure 43: Energy Function for Move Action

The Runaway action takes into nervousness and energy properties into consideration when choosing to run away from the player. Energy consideration uses an exponential decay function. This means that when energy is at the lowest, the utility score is close to 1 and when energy is from moderate to high, the utility score is close to 0. This ensures that one of the two reasons for Nue to run away is when energy is close to 0. Nervous consideration uses an exponential function with a weight of 0.8. So being nervous about the current situation makes the Nue want to run away. Combining with the energy, it makes sense that when energy is low and nervous is high, running away is the best possible scenario.

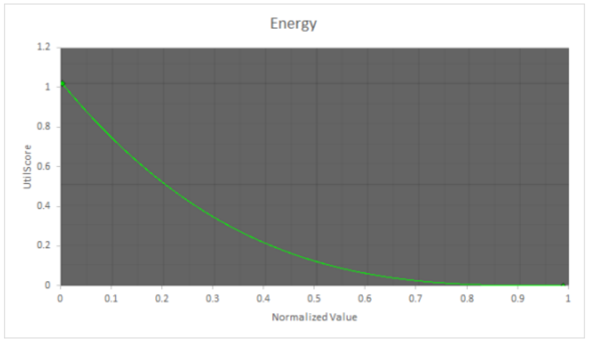


Figure 44: Energy Function for Run Away Action

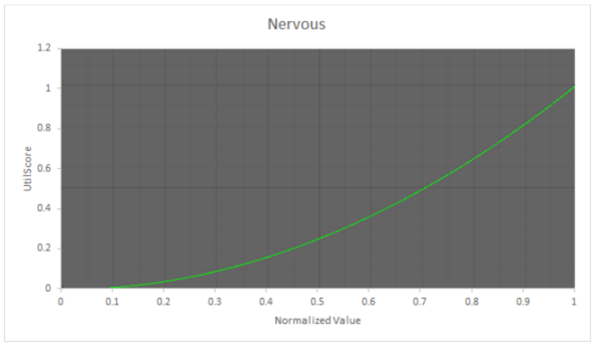


Figure 45: Nervous Function for Run Away Action

All of these functions work together in order to determine the best score. For example, in one scenario, Nue has a normalized anger value of 0.4, normalized calmness value of 0.1, normalized nervous value of 0.2 and a normalized energy value of 0.7. Figure 46 shows the calculations to determine the best action to take. The utility score for Charge action is 0.3933.

The utility score for Move action is 0.115. The utility score for Run Away action is 0.027. The highest utility score is Charge action so Nue execute the charging child node.

	Utility Value = PropertyUtil * Weight				Utility Formula: = Sum(PropertiesInvolved) / PropertiesInvolved
	Anger	Energy	Nervousness	Calmness	
Action(s)					
Charge	0.5	0.65	N/A	0.03	0.3933 (Highest Score)
Run Away	N/A	0.03	0.024	N/A	0.027
Move	N/A	0.2	N/A	0.03	0.115

Figure 46: Example of How the UtilityAI works

Let's look at another scenario where Nue has a normalized anger value of 0.4, normalized calmness value of 0.1, normalized nervous value of 0.2 and a normalized energy value of 0.2. Figure 47 demonstrates the calculation for this new scenario. The utility score for Charge action is 0.28. The utility score for Move action is 0.459. The utility score for Run Away action is = 0.262. In this new scenario, the highest utility score is Move action so Nue execute the move child node.

	Utility Value = PropertyUtil * Weight				Utility Formula: = Sum(PropertiesInvolved) / PropertiesInvolved
	Anger	Energy	Nervousness	Calmness	
Action(s)					
Charge	0.5	.003	N/A	0.03	0.28
Run Away	N/A	0.5	0.024	N/A	0.262
Move	N/A	0.9	N/A	0.018	0.459 (Highest Score)

Figure 47: Another Example of How the UtilityAI works

UtilityAI program does all these evaluations using its built-in function Evaluate() that loop through a list of all possible actions, evaluate the average utility score for each and then set the highest score in every loop until there are no more action to evaluate. The action's utility score is dependent on the utility curve of its considerations and the weight using the function EvaluateAction().

```
public float Evaluate(){
 BestScore = 0;
 BestAction = null;
 for each action a in a list of actions
  a.EvaluateAction();
  if a.Score() > BestScore then
   BestScore = a.Score();
   BestAction = a;

 return BestScore;
}
```

```
public void EvaluateAction(){
 score = 0;
 for each consideration c in a listOfConsiderations
  score = c.UtilityScore * c.Weight;

score = score/ listOfConsiderations.Count;

}
```

Code 16: Pseudocode of Evaluate() and EvaluateAction() working together in UtilityAI program

5.4.2 Tricky

Implementation Structure

Tricky personality is built on behavior tree. Behavior tree is a kind of implementation of AI. It determines which action the AI should take by running each child nodes from left to right and top to bottom. If one of the child nodes fails, which means it doesn't met certain condition, then it will return "fail" to its parent node and try to run the next child node of its parent. The relationship between parent nodes and child nodes are selector or sequence. Selector means the tree will success if any of its branch successes, while sequence needs all its child nodes or branches to succeed in order to make itself succeed [32].

The reason behavior tree is used is that we found a simple but effective way to beat player, which is keep charging as much as it can, and this feature can be implemented by behavior tree perfectly. We just need to make charge as the first rank action, move as the second rank action and melee attack as the third one. The structure of behavior tree made it so much simpler for us to do it.

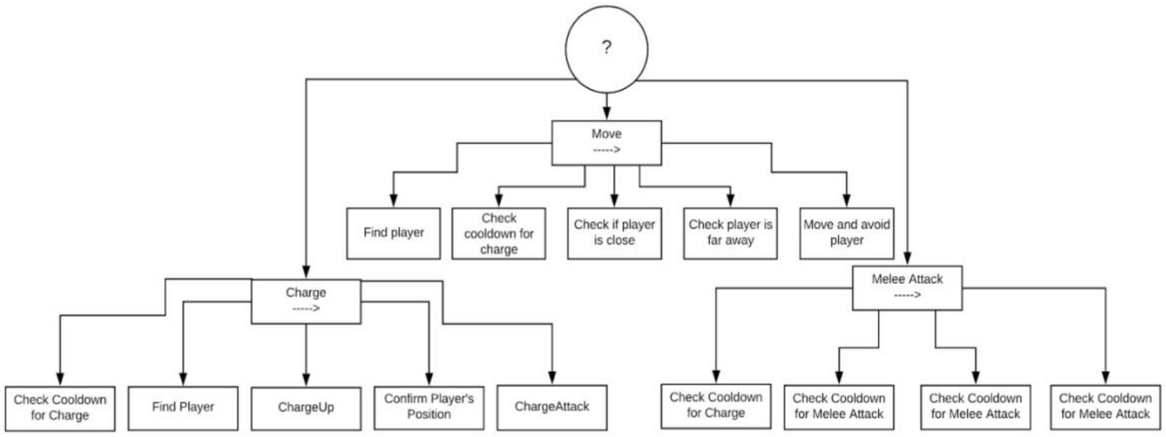

?
Move
----->
Find player
Check cooldown for charge
Check if player is close
Check player is far away
Move and avoid player
Charge
----->
Check Cooldown for Charge
Find Player
ChargeUp
Confirm Player's Position
ChargeAttack
Melee Attack
----->
Check Cooldown for Charge
Check Cooldown for Melee Attack
Check Cooldown for Melee Attack
Check Cooldown for Melee Attack

Figure 48: Tricky Behavior Tree

In Figure 48, it shows the behavior tree of tricky personality. This tree is a selector, which means it will be succeed if any of its sub nodes successfully run. The tree runs from left to right and top to bottom. The first action it will try to do is charge. It will attempt to charge and check if it can charge or not. If it can, then it will do the charge action. If it cannot, which means charge action is failed; it will start to do the moving action and then melee attack. If any of these nodes succeeds or all of them fail, the tree will be ended.

The pattern of this AI may look simple, yet follows an effective strategy. In this game, monster is invulnerable to player's attack only if it is in its charging state. Once it charges out, it won't receive any damage until it stops charging. The strategy of tricky personality is to utilize this mechanic as much as possible and to get away from players to avoid being attacked by them. There is a cooldown mechanic for it charging ability, which means it cannot use charging anytime it wants. While charging is still in cooldown, it needs to keep a distance with player in order to protect himself from player's attack, since it is not invulnerable while it's not charging. Note that its strategy is not to kill the player as soon as possible, but to keep itself alive and try to deal as much damage as possible to player. This is also the reason why players will never see monster with this tricky personality chase them around the map and do a lot melee attack. Monster moves way slower than players do, and even if it can catch a player by walking, the player will still be able to deal damage to it while it’s attacking player with its fist (melee attack). These are the reasons why it is a bad idea to try to catch player by walking and attacking them with melee attack action.

```
Tree Tricky
    repeat selector
        Charge
        Move
        Melee Attack
```

Code 17: Pseudo code for main branch of behavior tree Tricky

Note that this behavior tree will run over ten times in each frame, which means there is a function to make it repeat these behaviors. This function in our tree is called repeat, and it will repeatedly run certain behavior tree after they are ended. Code 17 shows how the tree is repeated in script.

Action - Charge

The first action to be considered in tricky personality behavior tree is “Charging”. As it is described in the game design section, charging is the most powerful ability of the monster. It let monster dash through a distance and deals a large amount of damage to players if it hits them. Another feature of charging action that makes it a powerful ability is that monster is invulnerable during the dash.

Powerful abilities, however, need to be limited otherwise player will not get a chance to beat this monster. In order to balance this ability, we added a cooldown mechanic to it.

```
protected void CheckChargeCooldown(){
    If charge cooldown is done {
        Keep going on charge action;
    }
    else{
        Move to the next branch;
    }
}

private void ChargeCooldownTimer(){
    Count time after charge ends;
}
```

Code 18: Pseudo code for charge cool down

As shown in Code 18, it is unique cooldown functions which count down a certain number of seconds and then flip the Boolean for cooldown to positive which indicates that cooldown for charging is finished. This function will be called when charging ends, and then it will start to count down certain numbers of seconds.

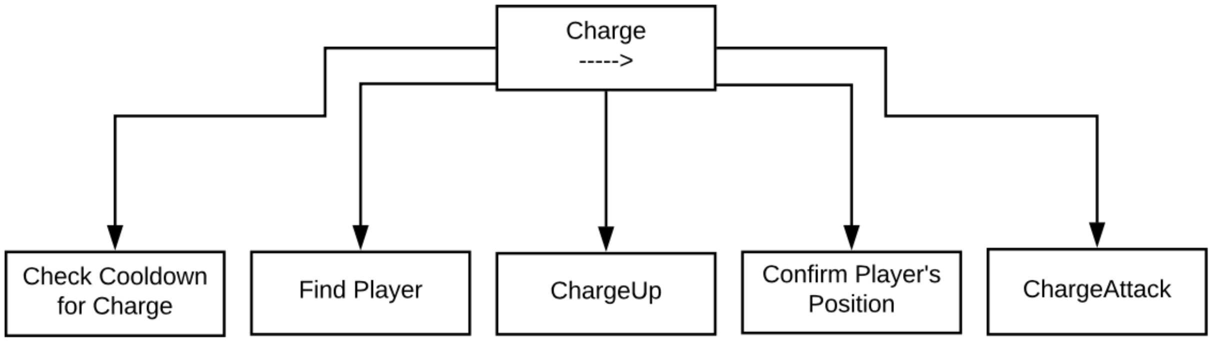


Figure 49: Charge Sequence Tree for Tricky Personality

In Figure 49, it shows the behavior tree branch for charge action. This is actually the most important branch among three branches, and it's the first rank action which is its cooldown will be checked in the very beginning. This is a sequence tree, which means all the child nodes have to succeed in order for the whole branch to succeed. If any of these child nodes fails, for example cooldown for charge is not ready, the whole branch will return fail to the main branch and the tree will decide to move to the next branch.

First, it will check for its cooldown to see if it is able to charge or not. If it is not able to charge then it will move to the next branch in the tree. If it is able to charge, however, the first thing it's going to do is to find the player.

```
protected bool FindPlayer(){
    Destination = player.getPostition();
    Return true;
}
```

Code 19: Pseudocode for finding player

As it is shown in Code 19, function "find player" will find the X and Y position of player, and then return true which indicates that the player is found. After the position of player is found, monster will start to charge up. Charge up, as mentioned in game design section, is a stage in charge which the monster will keep relocating the position of player and shaking its body to indicate that it's going to do charge attack.

```
protected void ChargeAttack(){
    If player is too far {
        MoveToNextBranch();
    }else{
    ChargeOut();
    StartCooldownTimer();
    }
}
```

Code 20: Pseudocode for ChargeAttack

After confirming the player's position for the one last time, the monster will charge out. Note that one big difference between tricky and other personalities is that it has its own rule of charging. First of all, it will not charge out if the player is not in a certain range. If the player is too far away from tricky, it will discard the charging action and starts the moving action instead.

The reason why it will do this is that it wants to get a bigger chance on hitting the player with charge. The farther players are, the more likely they will be able to dodge its charge attack, since charge attack is a linear attack, not a fan-shape one. This is also the reason why it will take in player's moving direction as a variable. We noticed that when the player is already moving, it is more difficult for the monster to hit them with charge attack. Therefore, we added more code in charge attack of tricky to make it adjust its charging direction if the player is currently moving.

Action - Move

The main goal of this action is to avoid being attacked by player.

In the beginning of this action, it will terminate all the functions that are still running for charge action, since charge is the action right before move. Then it will check if player is close enough for melee attack, if it is, go to the next branch which is melee attack.

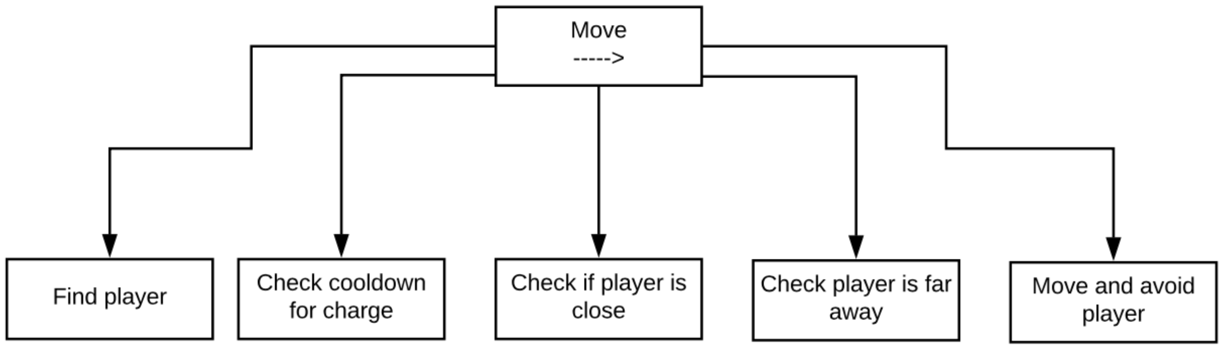


Figure 50: Move Sequence tree for Tricky Personality

Like it is mentioned in section for charge action, it will check for charging cooldown before it actually starts the moving action and then if the cooldown for charging is ready, it will move to the next branch as well. This time it will move until it reaches charging action again.

Finally, it will check if player is far enough from it. Since the main goal of this whole action is to avoid being attacked by player, as long as player is far enough from the monster, enough that player cannot reach monster, the monster would be safe and therefore stop moving. However, if player is not far away enough, the monster will start moving and try to avoid physical contact with the player.

```
protected void KeepDistance(){
    If player is in zone1 {
        AvoidPlayer(zone2);
    }
    ...
}
```

Code 21: Pseudo code for keeping distance from player

```
protected void AvoidPlayer(Zone zone){
    X = zone.x;
    Y = zone.y;
    Direction = (x - enemyPostion.x, y - enemyPostion.y);
    EnemyMove(Direction, zone);
}
```

Code 22: Pseudo code for Avoiding player

In function keep distance it divides the whole scene into four different sections: top-right, top-left, bottom-right and bottom-left. If player is in one of the section then the monster will start to move to another one until it is far enough from the player. In avoid player, it basically calculates for the specific direction for the monster to move. If player plays with tricky for a long period of time, they will be able to notice that there is a certain pattern in it motion, which is going escaping and going for corners. However, the charge action usually interrupts their observation and makes it more difficult for them to find its pattern.

Action - Melee Attack

The purpose of this action is to disperse players while it's charging action is still in cooldown. Since the strategy of tricky is to keep itself alive and attack player with charging action, melee attack become a backup plan for tricky to make itself survive from player's attack. It only does melee attack to the player if player is close enough and it is still not able to charge. If player try to dodge from its melee attack, it will be able to keep a larger distance with player and sustain longer before the charge attack cooldown ends.

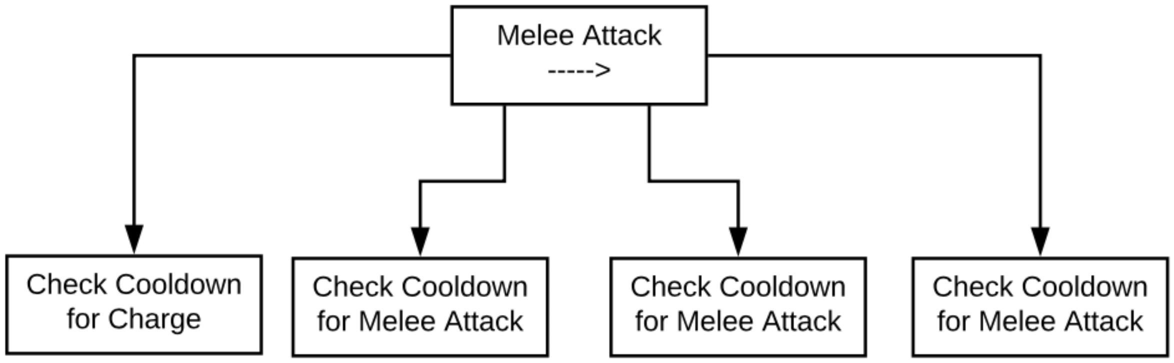

Melee Attack
----->
Check Cooldown for Charge
Check Cooldown for Melee Attack
Check Cooldown for Melee Attack
Check Cooldown for Melee Attack

Figure 51: Melee Sequence Tree for Tricky Personality

As shown in Figure 51, it will check for charge cooldown and if charge action is ready then it will end the tree, which means the tree will loop from charge action again.
If cooldown for charge is not ready, tricky will find the player and melee attack the direction where the player is in.

```
protected bool MeleeAttackWithCooldown(){
    If attack cooldown is ready{
        attack();
    }
}
```

Code 23: Pseudo code for Melee attack

```
protected void cooldownTimer(){
    Count time after attack ends;
}
```

Code 24: Pseudo code for attack cool down

Note that melee attack has a cooldown mechanic just like charge attack, and the two cooldown functions share the same logic as well.

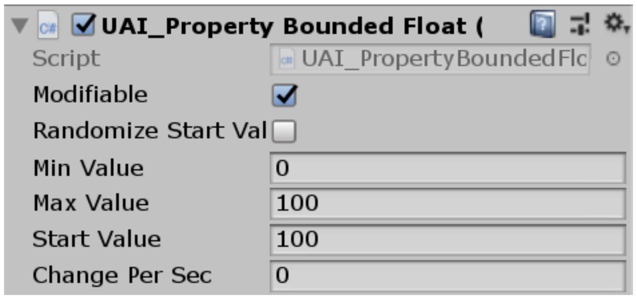
5.4.3 Bullying

The bullying personality is built with utility based decision maker (UtilityAI) and behavior tree (Panda BT), same as the aggressive personality. It is supposed to make the Yokai aggressive and bully the player when it feels like the player is weak and exploitable, but become afraid and start to run away when it feels like the player is strong.

Properties

In order to create this behavior, at least two properties are required:

Anger



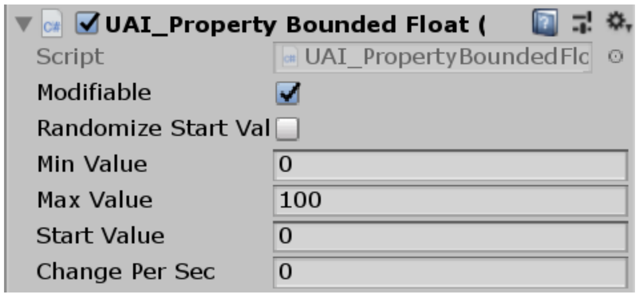
UAI_Property Bounded Float (

Script	UAI_PropertyBoundedFlc
Modifiable	✓
Randomize Start Val	☐
Min Value	0
Max Value	100
Start Value	100
Change Per Sec	0

Figure 52: Anger property

Anger determines how aggressive Nue is when bullying the player. It ranges from 0 to 100, and starts at 100. The reason for it to start at 100 is that we want it to aggressively bully the player at the start.

Nervousness


UAI_Property Bounded Float (

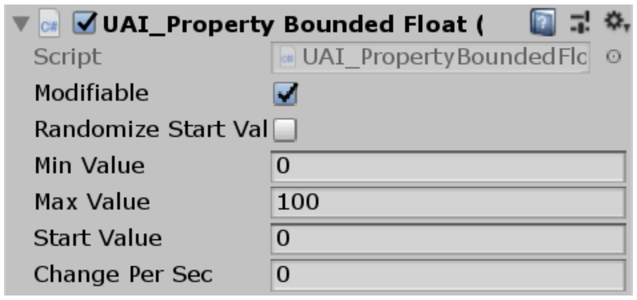
Script	UAI_PropertyBoundedFlc
Modifiable	✓
Randomize Start Val	☐
Min Value	0
Max Value	100
Start Value	0
Change Per Sec	0

Figure 53: Nervousness property

Nervousness is how afraid Nue is and how willing it is to actively attack. It ranges from 0 to 100, and starts at 0, because it should not be afraid of the player when it doesn't even know how strong the player is.

Two additional properties are created for better gameplay experience:

Tiredness

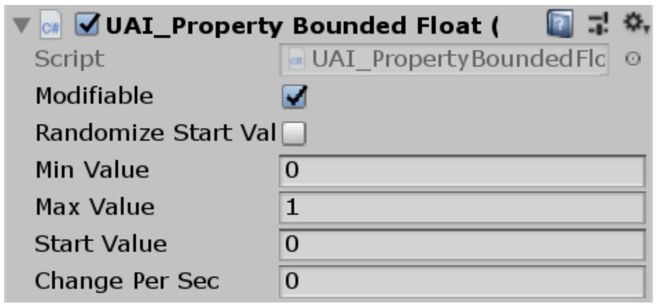

UAI_Property Bounded Float (

Script	UAI_PropertyBoundedFlc
Modifiable	✓
Randomize Start Val	☐
Min Value	0
Max Value	100
Start Value	0
Change Per Sec	0

Figure 54: Tiredness property

Tiredness determines how tired the Yokai is. It ranges from 0 to 100, and starts at 0. When it is tired, it is less aggressive so player can get some rest and adjust his/her positioning.

Player close enough



UAI_Property Bounded Float (

Script	UAI_PropertyBoundedFlc
Modifiable	✓
Randomize Start Val	☐
Min Value	0
Max Value	1
Start Value	0
Change Per Sec	0

Figure 55: PlayerCloseEnough property

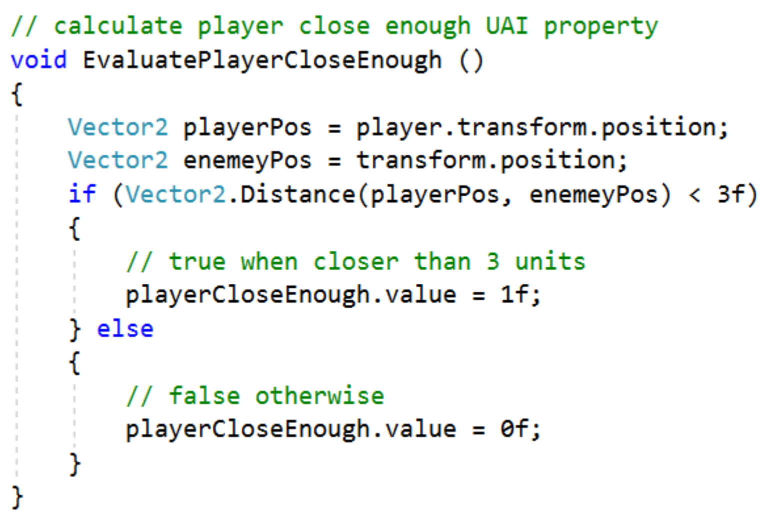
Player close enough determines whether player is in range of Nue's melee attack. It is either 0 or 1, while 0 means close enough and 1 means not close enough. It functions as a Boolean property, but since all others are floating point properties, it is easier to process by making it a floating point property as well.

```
// calculate UAI properties based on health
void EvaluateHealth()
{
    if (bossCurrentHealth <= lastCheckHealth - 10)
    {
        Debug.Log("health--");
        nervous.value += 5;
        lastCheckHealth = bossCurrentHealth;
    }

    if (bossCurrentHealth < player.playerCurrentHealth)
    {
        if (!isSetRetreat) {
            preRetreatNervousValue = nervous.value;
            isSetRetreat = true;
        }
        nervous.value = 100;
    } else {
        if (isSetRetreat) {
            isSetRetreat = false;
            nervous.value = preRetreatNervousValue;
        }
    }
}
```

Code 25: Evaluate health

Change of Properties

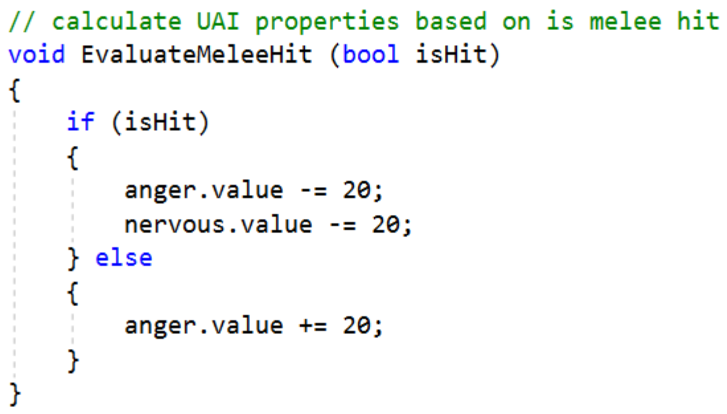
For every frame, EvaluateHealth() is called and Nervousness value is changed. When Nue has 10 HP less than previously checked, Nervousness increases by 5. When Nue's health is lower than player's, current Nervousness value is cached and set to 100. When Nue's health is higher than player's after being lower than player's, current Nervousness value is reset to previously cached value. Resetting Nervousness value allows Nue to regain confidence after regaining health advantage.



```
// calculate player close enough UAI property
void EvaluatePlayerCloseEnough ()
{
    Vector2 playerPos = player.transform.position;
    Vector2 enemeyPos = transform.position;
    if (Vector2.Distance(playerPos, enemeyPos) < 3f)
    {
        // true when closer than 3 units
        playerCloseEnough.value = 1f;
    } else
    {
        // false otherwise
        playerCloseEnough.value = 0f;
    }
}
```

Code 26: Evaluate player close enough

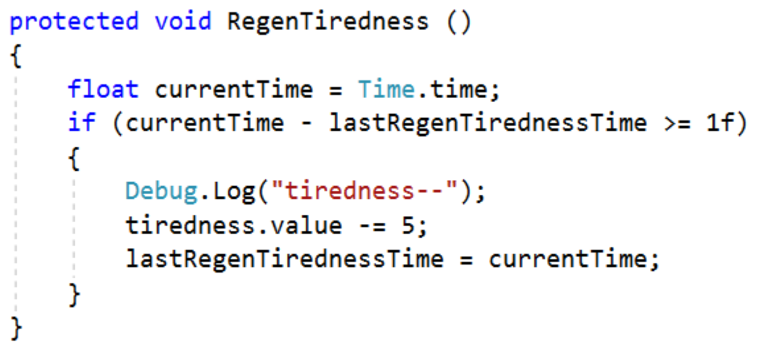
For every frame, EvaluatePlayerCloseEnough() is called and Player Close Enough value is changed. If the player is within 3 units, Player Close Enough is set to 1, if not, Player Close Enough is set to 0.



```
// calculate UAI properties based on is melee hit
void EvaluateMeleeHit (bool isHit)
{
    if (isHit)
    {
        anger.value -= 20;
        nervous.value -= 20;
    } else
    {
        anger.value += 20;
    }
}
```

Code 27: Evaluate melee hit

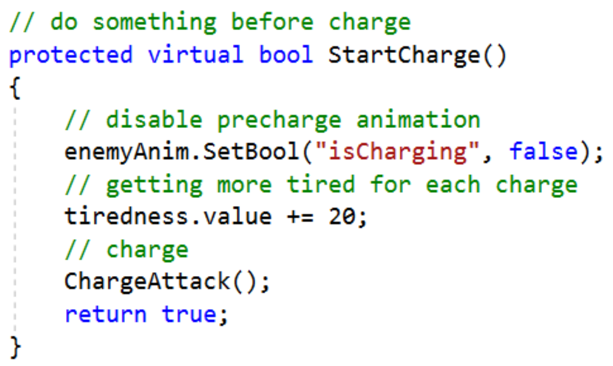
Whenever a melee attack or charge finishes, EvaluateMeleeHit() or EvaluateChargeHit() is called, Anger and Nervousness values are modified. When melee or charge is hit, Anger is reduced by 20 and Nervousness is reduced by 20, if missed, Anger is increased by 20. This means that if the player is hit multiple times in a row, Nue will not be as angry and eager to kill the player, but relax and take more time to enjoy the hunt. However, once it starts missing hits, it gets angrier because the player is causing more trouble than it expected.



```
protected void RegenTiredness ()
{
    float currentTime = Time.time;
    if (currentTime - lastRegenTirednessTime >= 1f)
    {
        Debug.Log("tiredness--");
        tiredness.value -= 5;
        lastRegenTirednessTime = currentTime;
    }
}
```

Code 28: Tiredness decrease

When Nue is in Walk mode, RegenTiredness() is called every frame, and Tiredness



```
// do something before charge
protected virtual bool StartCharge()
{
    // disable precharge animation
    enemyAnim.SetBool("isCharging", false);
    // getting more tired for each charge
    tiredness.value += 20;
    // charge
    ChargeAttack();
    return true;
}
```

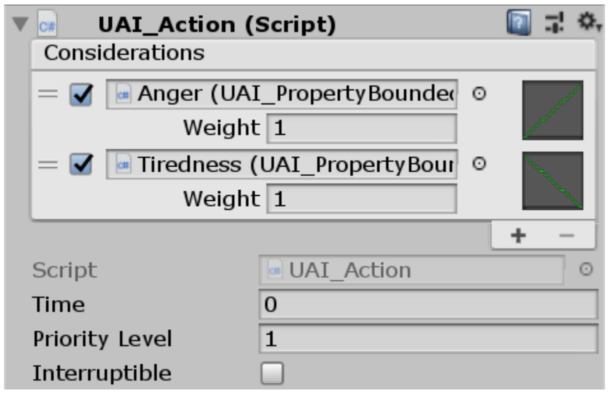
value decreases by 5 every second.

Code 29: Tiredness increase

Whenever Nue charges, Tiredness value increases by 20.

Actions and Utility Functions

Charge



UAI_Action (Script)

Considerations

Anger (UAI_PropertyBounde	Weight 1
Tiredness (UAI_PropertyBour	Weight 1

Script	UAI_Action
Time	0
Priority Level	1
Interruptible	☐

Figure 56: Charge action

Utility score of Charge is determined by Anger and Tiredness, each weighing 1.

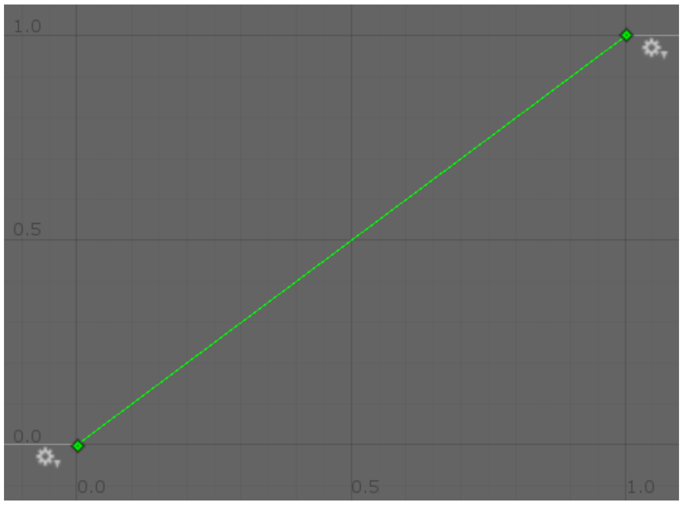


Figure 57: Charge's relationship with Anger

It has a positive relationship with Anger, with utility function y=x. So the angrier Nue is, the higher priority Charge has.

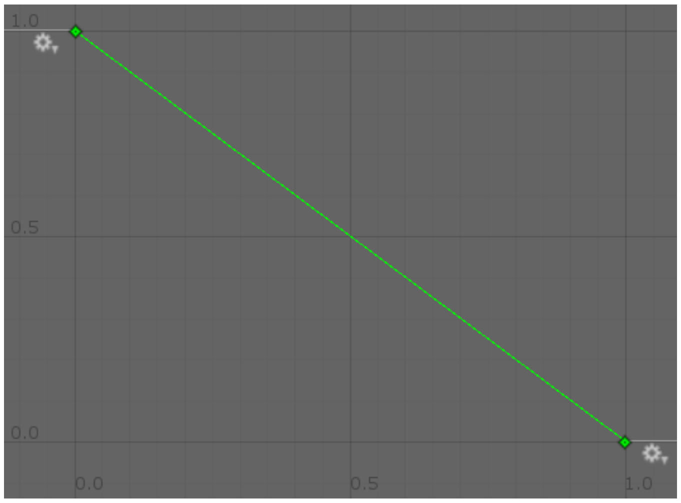
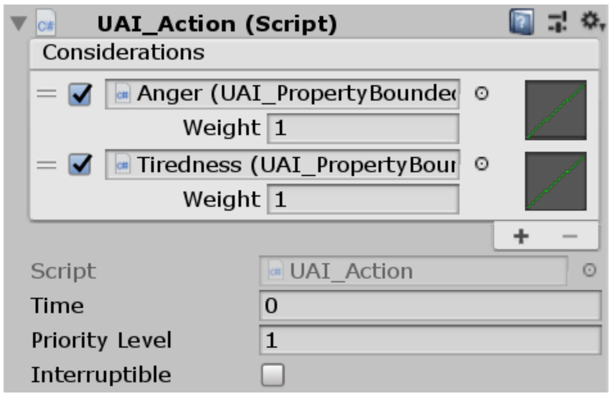


Figure 58: Charge's relationship with Tiredness

It has a negative relationship with Tiredness, with utility function y=-x. So the more tired Nue is, the lower priority Charge has.

Walk



UAI_Action (Script)

Considerations

Anger (UAI_PropertyBoundeα
Weight 1

Tiredness (UAI_PropertyBour
Weight 1

Script	UAI_Action
Time	0
Priority Level	1
Interruptible	☐

Figure 59: Walk action

Utility score of Walk is also determined by Anger and Tiredness, each weighing 1.

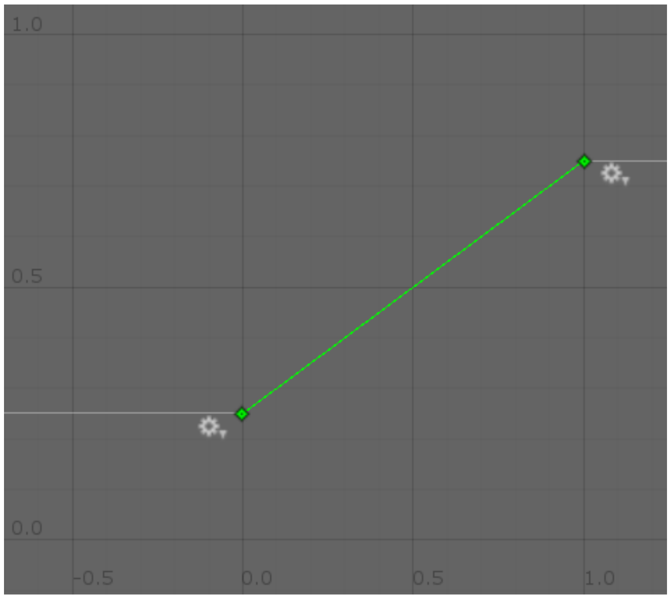


Figure 60: Walk's relationship with Anger

It has a positive relationship with Anger with utility function y=0.5x+0.25. It still means that the angrier Nue is, the higher priority Walk has compared to Actions other than Charge. However, assuming Tiredness provides the same utility score, Charge has a higher priority when Anger is higher, and Walk has a higher priority when Anger is lower.

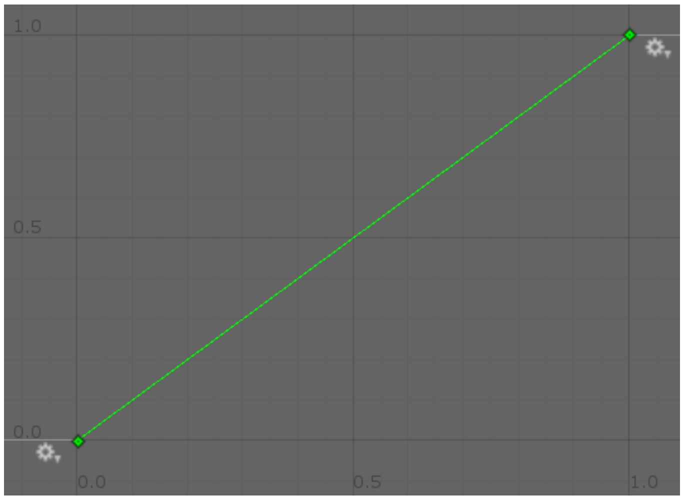
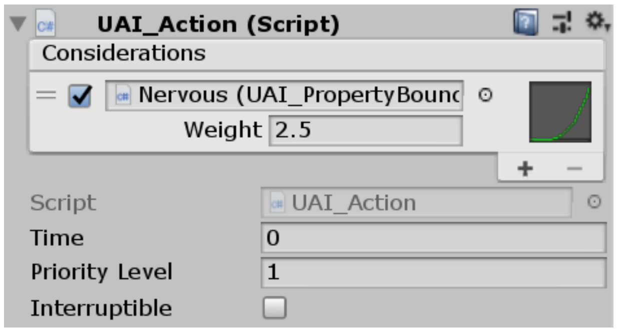


Figure 61: Walk's relationship with Tiredness

It has a positive relationship with Tiredness, with utility function y=x. So the more tired Nue is, the higher priority Walk has.

Retreat


UAI_Action (Script)

Considerations

Nervous (UAI_PropertyBounc

Weight 2.5

Script UAI_Action

Time 0

Priority Level 1

Interruptible

Figure 62: Retreat action

Utility score of Retreat is only determined by Nervousness, weighing 2.5. The reason for it to weigh 2.5 is to allow it to have higher maximum utility score (2.5) than Walk or Charge (2.0), so Nue will always retreat when Nervousness is high, ignoring other actions.

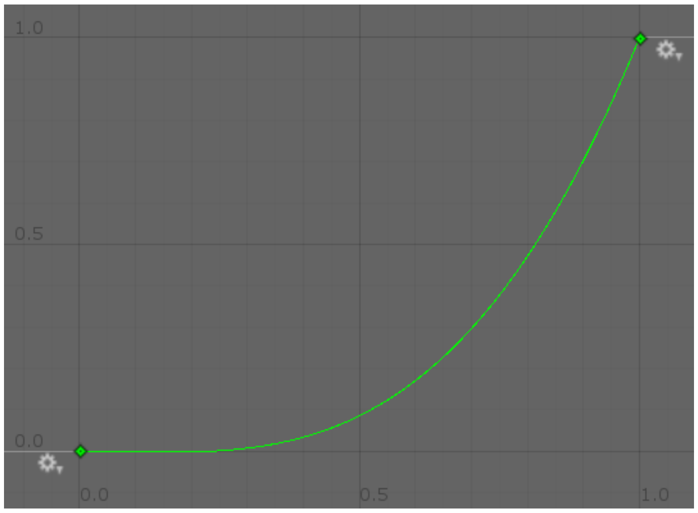
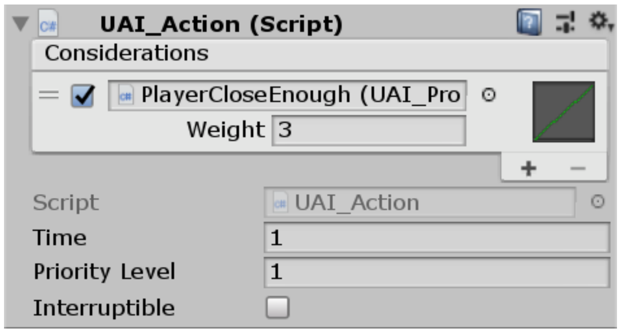


Figure 63: Retreat's relationship with Nervousness

It has a curved positive relationship with Nervousness, with utility function approximately y=x^2. The reason to make it a curve, is to guarantee its retreat when Nervousness is extremely high, while prevent it from activating too early when Nervousness is still low, due to its higher weight.

Melee


UAI_Action (Script)

Considerations

PlayerCloseEnough (UAI_Pro

Weight 3

Script UAI_Action

Time 1

Priority Level 1

Interruptible

Figure 64: Melee action

Utility score of Melee is only determined by Nervousness, weighing 3. The reason for it to weigh 3 is to allow it to have higher maximum utility score (3) than any other actions, so Nue will always melee when Player Close Enough is 1, ignoring other actions.

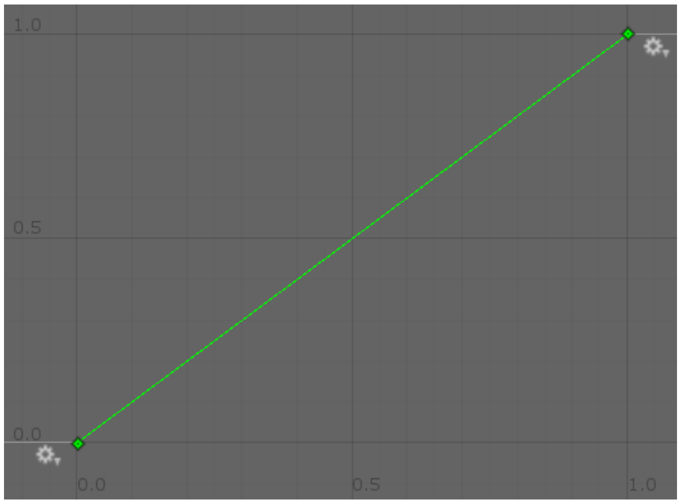


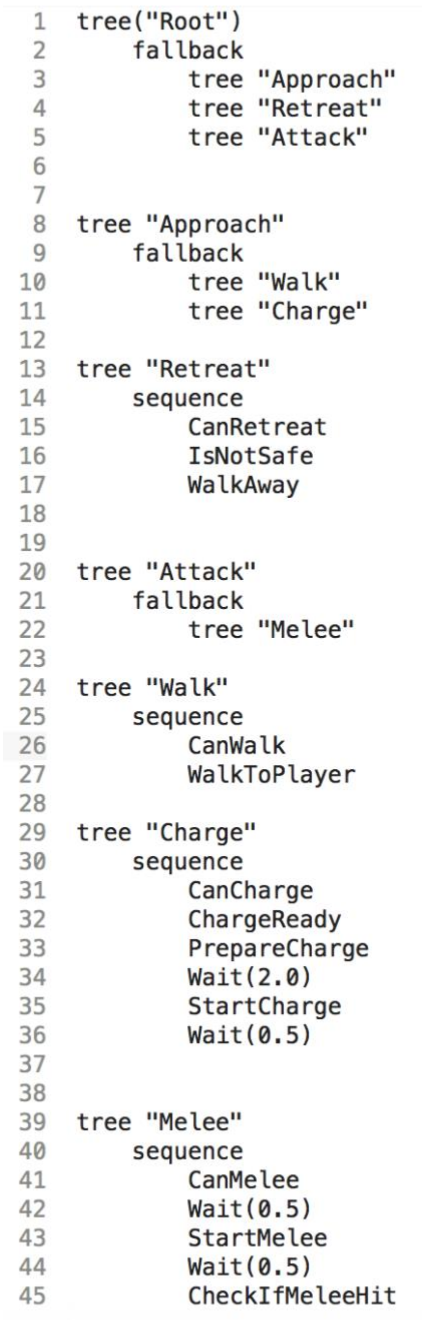
Figure 65: Melee's relationship with PlayerCloseEnough

It has a positive relationship with Player Close Enough, the utility function is y=x. However, it functions as a discrete function, since Player Close Enough will only be either 0 or 1, the utility score of Melee will only be either 0 or 3.

Behavior Tree

The behavior tree is run repeatedly from the Root node. Nodes with “fallback” in script, and “?” on diagram are selector nodes, nodes with “sequence” in script, and “->” on diagram are sequence nodes. Every sequence node has a gate, named “Can + action”, and only 1 gate will be open for each run, so only 1 action will be running at a time. A gate opens only when the current state (set by UtilityAI) is the gate’s action, the action is not in cooldown and there isn’t any ongoing action.

The tree used for Bullying has 4 action sequence nodes: Retreat, Melee, Charge and Walk. Below is the raw text script for Panda BT.



```
tree("Root")
    fallback
        tree "Approach"
        tree "Retreat"
        tree "Attack"


tree "Approach"
    fallback
        tree "Walk"
        tree "Charge"

tree "Retreat"
    sequence
        CanRetreat
        IsNotSafe
        WalkAway


tree "Attack"
    fallback
        tree "Melee"

tree "Walk"
    sequence
        CanWalk
        WalkToPlayer

tree "Charge"
    sequence
        CanCharge
        ChargeReady
        PrepareCharge
        Wait(2.0)
        StartCharge
        Wait(0.5)


tree "Melee"
    sequence
        CanMelee
        Wait(0.5)
        StartMelee
        Wait(0.5)
        CheckIfMeleeHit
```

Code 30: Panda BT Script of a Behavior Tree for Bullying

Code 30 can be translated to the diagrams below.

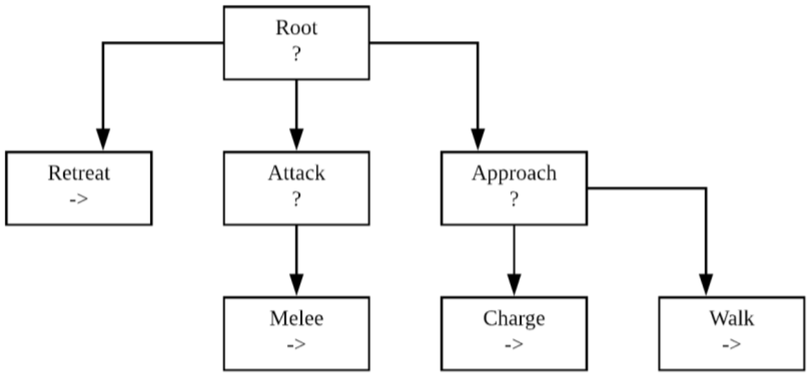


Figure 66: Behavior tree

Retreat

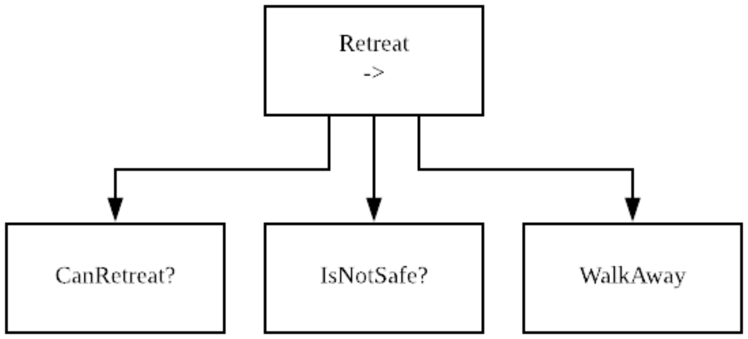


Figure 67: Retreat sequence node

CanRetreat is the gate for Retreat. IsNotSafe also needs to be satisfied to execute WalkAway, but it is not controlled by UtilityAI.

Melee

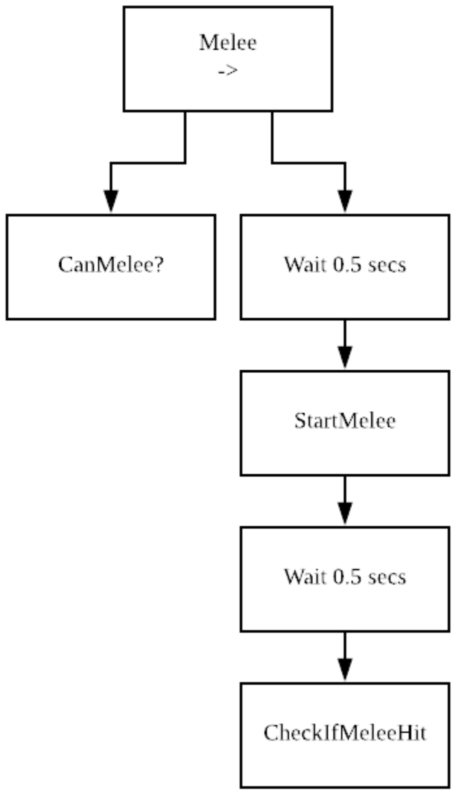


Figure 68: Melee sequence node

CanMelee is the gate for Melee. Once it is satisfied, it will wait for 0.5 second before StartMelee for player to dodge. After StartMelee, it will wait for an additional 0.5 second and check if the melee attack hits or not.

Charge

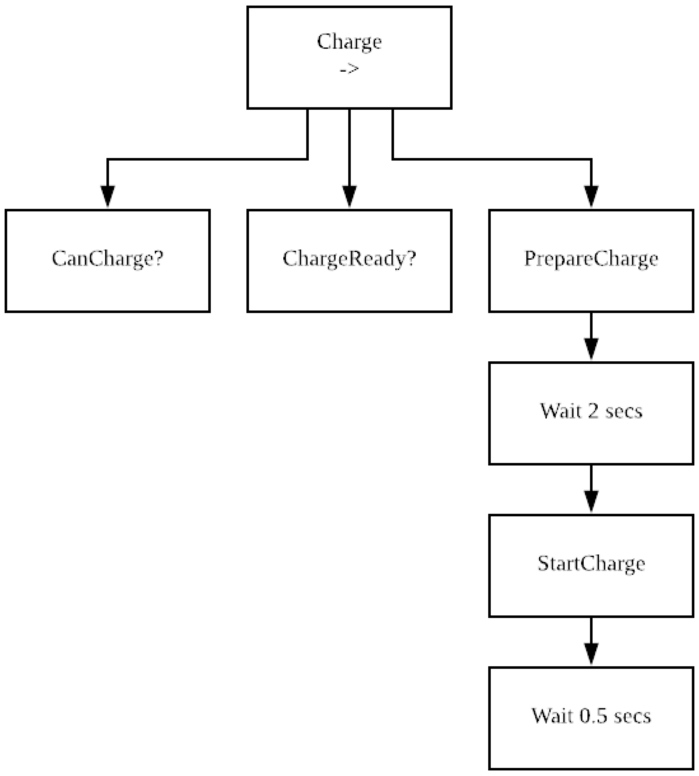


Figure 69: Charge sequence node

CanCharge is the gate for Charge. ChargeReady also needs to be satisfied to run PrepareCharge, which plays the pre-charge animation for 2 seconds, until StartCharge. Finally it waits for 0.5 second for Nue to hit.

Walk

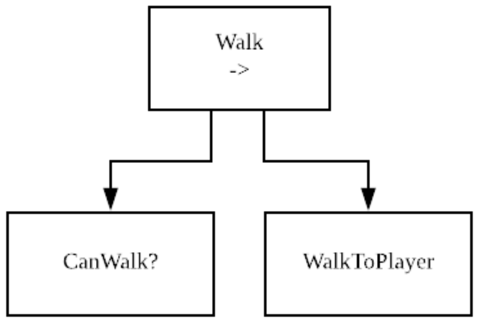


Figure 70: Walk sequence node

CanWalk is the gate for Walk. Once it is satisfied, WalkToPlayer will be executed.

5.4.4 Cowardly

The cowardly personality is built as a state machine. It is supposed to make Nue to be always retreating, always running away, but at the same time, be as annoying and difficult to defeat as other more aggressive personalities.

States

The state machine has 4 states: Retreat, Attack, Charge and Walk.

Retreat

Action sequence

1. Walk away from the player

Switch state conditions

- If player closer than 6 units and charge is ready -> switch to Charge
- Else if player closer than 3 units -> switch to Melee
- Else if player farther than 6 units for more than 3 seconds -> switch to Walk

Melee

Action sequence

1. Wait for 0.5 second
2. Melee attack
3. Wait for 0.5 second

Switch state conditions

- If charge is ready -> switch to Charge
- Else if player farther than 3 units -> switch to Retreat

Charge

Action sequence

1. Wait for 2 seconds
2. Charge away from the player / if cornered or colliding with a wall, charge alongside the wall
3. Wait for 0.5 second

Switch state conditions

- If player is closer than 3 units -> switch to Melee
- Else if player is closer than 6 units -> switch to Retreat
- Else if player is farther than 6 units -> switch to Walk

Walk

Action sequence

1. Walk towards the player

Switch state conditions

- If player is closer than 6 units -> switch to Retreat

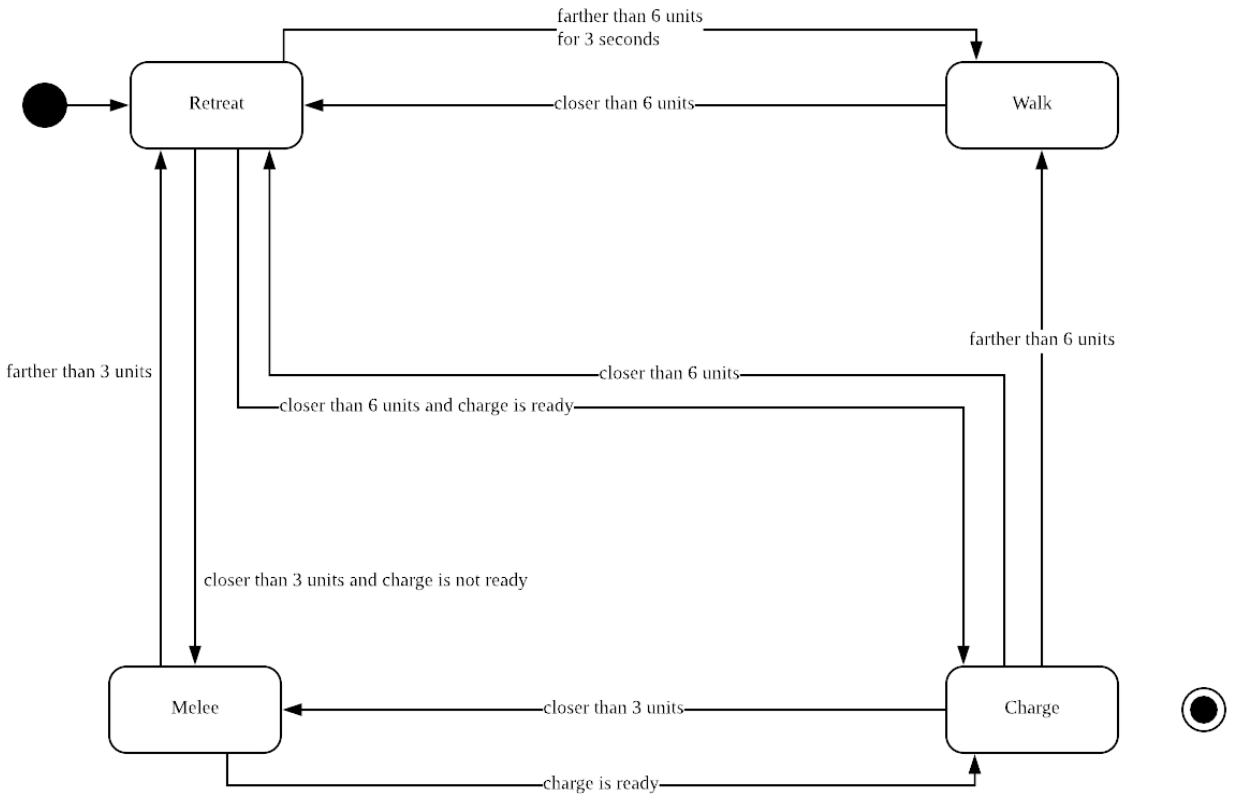


Figure 71: State diagram

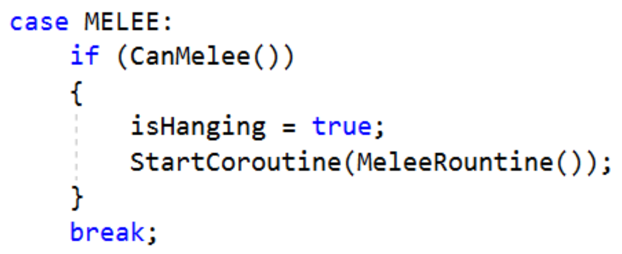
Combining all the conditions above:

1. Nue will always retreat when the player is too close (< 6 units), unless conditions for charge or melee is satisfied.
2. Nue will always melee when the player is within range (< 3 units), unless conditions for charge is satisfied.
3. If player is continuously chasing Nue and charge is ready, it will charge away to create more distance.

4. If charge is ready after melee attack, it will charge away regardless of whether its attack hits or not. Charge has an 8 second cooldown, preventing it from charging too often.
5. Nue will walk back to the player if player is not close (> 6 units) after charge, because it will think that it is safe. It will also walk back to the player if player is not close (> 6 units) for 3 seconds when retreating, before switching its state to Retreat because it walks too close. This means it is always trying to keep itself at its safety zone.

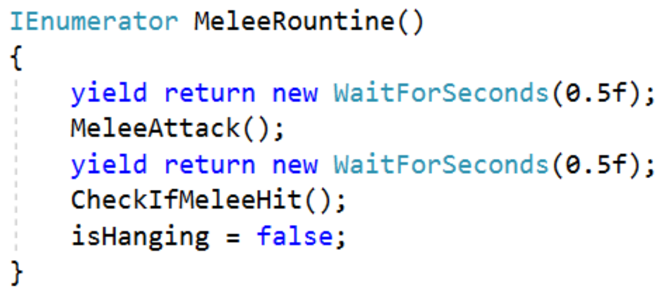
Implementation Structure

Because there are delays before and after melee and charge, Coroutine is used together with a *hanging* flag.



```
case MELEE:
    if (CanMelee())
    {
        isHanging = true;
        StartCoroutine(MeleeRountine());
    }
    break;
```

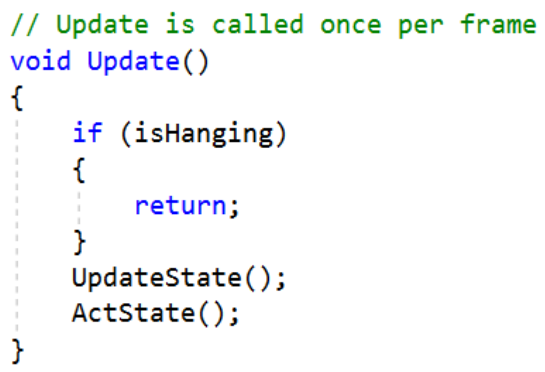
Code 31: Setting hanging flag



```
IEnumerator MeleeRountine()
{
    yield return new WaitForSeconds(0.5f);
    MeleeAttack();
    yield return new WaitForSeconds(0.5f);
    CheckIfMeleeHit();
    isHanging = false;
}
```

Code 32: Resetting hanging flag

When a Coroutine starts, *hanging* flag is set, when a Coroutine finishes, *hanging* flag is unset. The state machine will not switch state or perform any action when *hanging* flag is set, preventing mismatch between current action and current state, or start a new action while current action is still in progress.



```
// Update is called once per frame
void Update()
{
    if (isHanging)
    {
        return;
    }
    UpdateState();
    ActState();
}
```

Code 33: Sequence for every frame

So for every frame:

1. If *hanging* flag is set, return immediately
2. Switch state if any state switching condition is met
3. Perform actions of current state

Chapter 6: Testing and Result

6.1 Overview

In this chapter, we describe the methodology of our playtest design and the results. The purpose of the playtest is to evaluate how well players can recognize the personalities of our AI through four sessions of playing against it. The sections in this chapter are as followed: the method, the survey, result and finally the analysis.

6.2 Population

The sample population of our playtest design is everyone. Since our game has some element of Japanese culture incorporated, we wanted to know how well do the participants know about the existence of Nue in Japanese folklore. However, the most important data we want to collect is about how well participants can recognize the personalities of Nue through the gameplay. Having basic understanding of personalities is a must for one to give us valuable feedbacks. Testing these two elements in our game is important in determining how well our AI personalities are model.

6.3 Methodology

The procedure of our play test session is very simple. After participants have read and sign the consent form, we provided a thorough instruction form for the playtesters to read before playing the game. We ran our playtest for the week of 9/24/18 and the slots for the week was from 10 AM to 6 PM. The location of our play test is in one of the sponsored lab at Ritsumeikan University, CM lab.

There three parts to the playtest sessions. The first part of a session is to fill out a pre-survey.

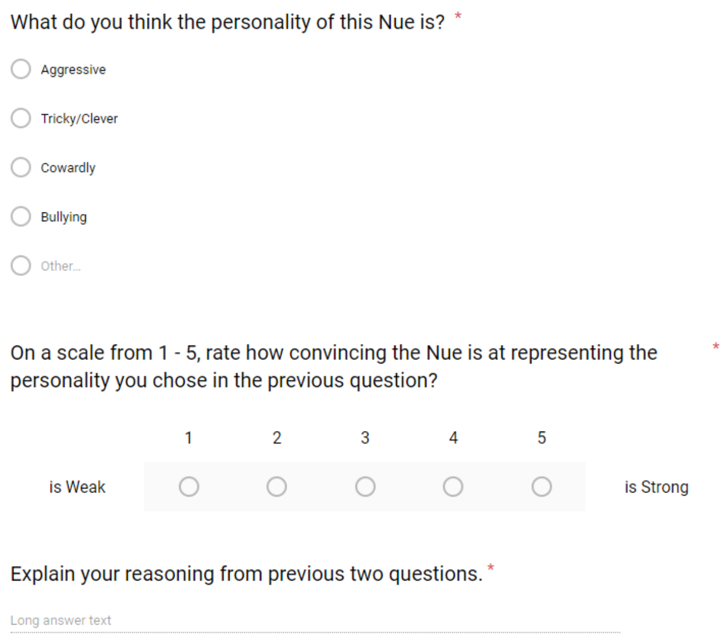
The second part of the session is to have the playtesters play our game. Once the game starts, we, the investigators, leave the playtesters alone to play the game. There is a tutorial and sound effects in the game that we were confident in guiding the player forward so we did not need to be in the presence while the game starts. There are four rounds of the game, each with different personalities. We did not provide the playtesters the order of the personalities or whether there are duplicates.

After each round, the playtesters would answer section of the survey that correspond to the round before moving on to the next. We also allowed playtesters the ability to play the round again as many times as they wanted. This allow players an opportunity to get a second opinion on the personality before submitting their responses. To help make this process as efficient as possible, we added a cutscene and a notification screen to notify the player to fill out the survey. To ensure instructions were followed, we check in with the play testers from time to time.

The final part of the playtest session is the post-survey. After the playtester finished the game, they were instructed to filled the post survey.

6.4 Survey

All of our data are collected through surveys. The first part of the survey is for gauging the player testers' knowledge of our boss character Nue. The second part of the survey is about collecting the playtesters thoughts and opinion on the Nue personality in each round. Figure 6.4.1 detailed the questions that are in the survey for this section. The final part of the survey is about understanding the playtesters overall impression of our game. We represent the data received in bar charts and pie charts.


What do you think the personality of this Nue is? *

○ Aggressive
○ Tricky/Clever
○ Cowardly
○ Bullying
○ Other...

On a scale from 1 - 5, rate how convincing the Nue is at representing the personality you chose in the previous question? *

	1	2	3	4	5	
is Weak	○	○	○	○	○	is Strong

Explain your reasoning from previous two questions. *

Long answer text

Figure 72: Second Section Survey Questions For Each Round

6.5 Challenges

One of the challenges we faced during our testing sessions was that our playtesters would skim the instructions and often play the game without stopping to fill out the survey before moving on. We decided it was best for us to check in with them once every 5 minutes, which was approximately how long each round of the battle is, to make sure the survey was filled.

The second challenge was that about half of our playtesters do not have a full grasp of English as their secondary or third language. Thankfully, we had members in our group and our lab that were willing to explain the process to the playtesters in Chinese and/or Japanese. When there was not an opportunity to translate, we had to use gestures and limit the speaking to one or two words to make it easier to understand. In addition, we observed that a few of our playtesters used google translate to translate the questions in order to understand it. We also allowed the playtesters to write in their most comfortable language and we will do our very best to translate it the best of our abilities.

6.6 Result and Analysis

Before getting into the details, a total of 12 responses by 3 subjects on the question “On a scale from 1 - 5, rate how convincing...” for each round are marked as outliers, and will not be counted in the analysis. They are marked yellow in the result section. The reason for this is three folds.

First, the response from them on the convincing rating, especially on the Cowardly round, is significantly deviated from all other responses. All three of them gave a 1 of out 5 for how confident they are when saying the A.I. in the third round is Cowardly, while the majority of others gave either a 4 or 5.

Second, the explanation of why they gave the convincing rating is questionable. The first subject with ID 4 said the reason he/she thinks the A.I. in round 1 is Aggressive is “it’s tried to kill me”, without giving any explanation of why he/she is only 3 out of 5 confident with the choice. The first subject also said the reason he/she thinks the A.I. in round 3 is “easier than before” but is not at all confident is “just stay”, which doesn’t contain any meaningful information. The second subject with ID 5 only gave objective opinions like “it is cute”, “it is cool”, “it is not good enough” and “it is fast”. The third subject with ID 6, his/her explanations contradict with his/her convincing rating. For example, the reason for him/her to give a 1 out of 5 in round 3 for his confidence of picking Cowardly is “Because the Nue appropriately runs away from the player by dashing.”, which means he/she is very convinced as he/she can relate the behavior of dashing away with Cowardly, but still gave a 1 out of 5.

Third, two of the three subjects used Japanese as response language, while the other one is very likely a non-native English speaker due to the grammar and the short response.

Combining the facts together, it is highly likely the three subjects misunderstood the convincing rating question as “how good do you think the A.I. is” rating question. Therefore their convincing ratings will be counted as outliers, and will not be used in data analysis.

6.7.1 Recognizability

For round 1 which uses Aggressive A.I., 68.8% of subjects correctly recognized the A.I. is Aggressive. The average convincing rating is 3.78 (only for subjects who got it right). Out of 9 subjects and from 1 to 5, 2 of them gave a 5, 5 of them gave a 4, 2 of them gave a 2.

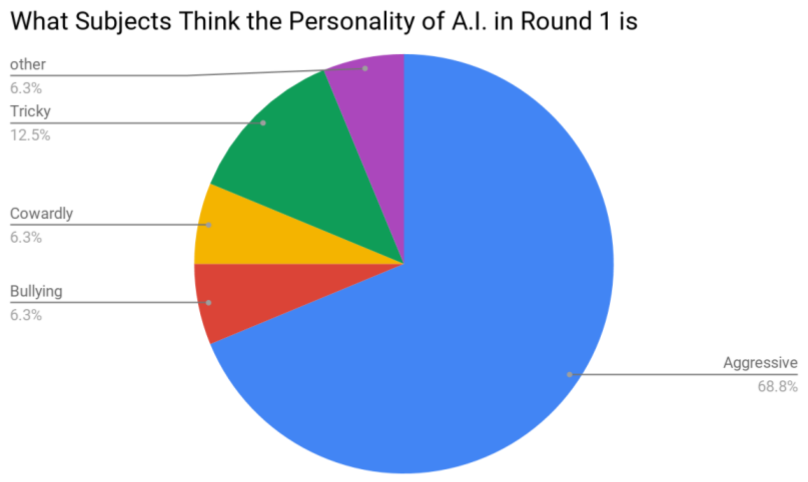


Figure 73: Subject’s opinion on what is the personality of A.I. in Round 1

For round 2 which use Bullying A.I., 12.5% of subjects correctly recognized the A.I. is Bullying. The average convincing rating is 3.50 (only for subjects who got it right). Out of 2 subjects and from 1 to 5, 1 of them gave a 3, 1 of them gave a 4.

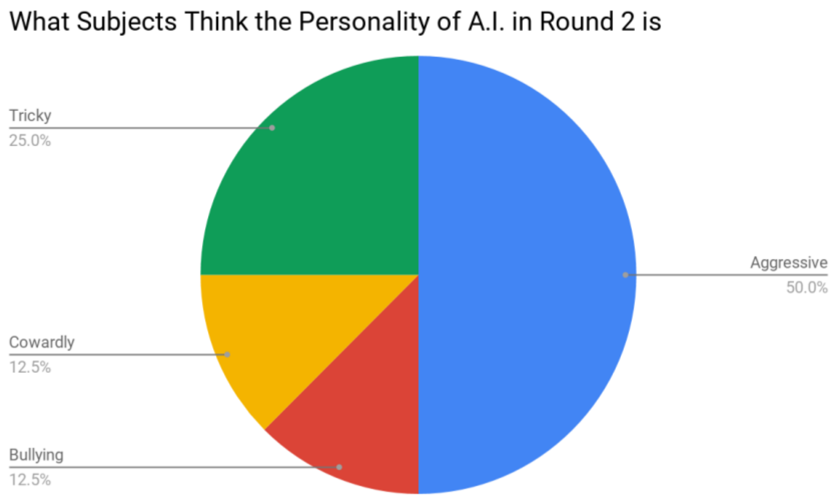


Figure 74: Subject's opinion on what is the personality of A.I. in Round 2

For round 3 which use Cowardly A.I., 75.0% of subjects correctly recognized the A.I. is Cowardly. The average convincing rating is 4.70 (only for subjects who got it right). Out of 10 subjects and from 1 to 5, 7 of them gave a 5, 3 of them gave a 4.

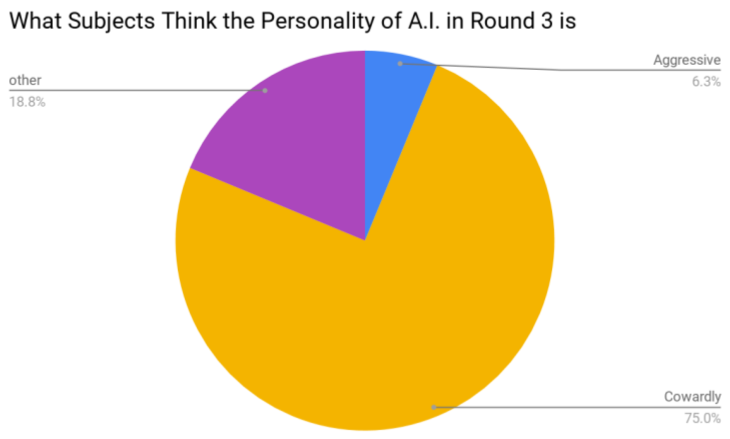


Figure 6.7.3:

Figure 75: Subject's opinion on what is the personality of A.I. in Round 3

For round 4 which use Tricky A.I., 62.5% of subjects correctly recognized the A.I. is Tricky. The average convincing rating is 4.38 (only for subjects who got it right). Out of 8 subjects and from 1 to 5, 4 of them gave a 5, 3 of them gave a 4, 1 of them gave a 3.

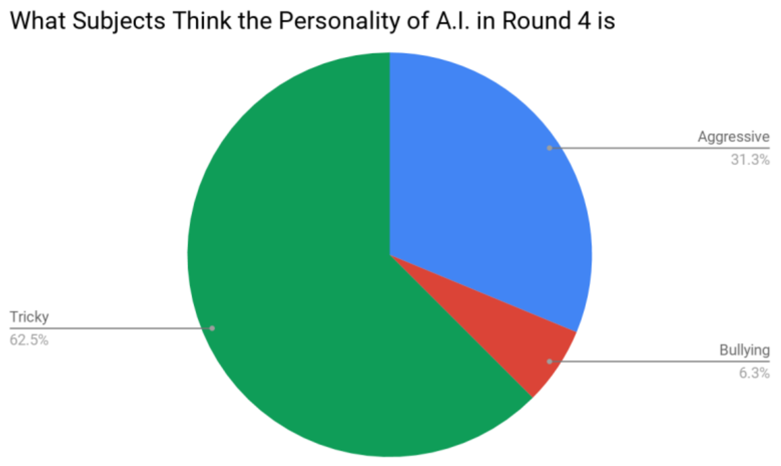


Figure 76: Subject’s opinion on what is the personality of A.I. in Round 4

In order to evaluate which A.I. is the most recognizable, a Recognizability score is made with function:

Recognizability =

Percentage of subjects correctly recognizing an A.I.’s personality

×

Average convincing rating of this A.I. by subjects who got it right

Therefore:

Recognizability score for Aggressive is 2.60.

Recognizability score for Bullying is 0.44.

Recognizability score for Cowardly is 3.53.

Recognizability score for Tricky is 2.74.

From this we can see that Cowardly is the most recognizable, Tricky being the second, followed closely by Aggressive, and Bullying being not at all recognizable.

6.7.1 Believability

For which A.I. is the most believable:

50% of all subjects think Aggressive A.I. best resembles what they think Nue is.

6.3% of all subjects think Bullying A.I. best resembles what they think Nue is.

0% of all subjects think Cowardly A.I. best resembles what they think Nue is.

25% of all subjects think Tricky A.I. best resembles what they think Nue is.

18.7% of all subjects think None of the 4 A.I.s resembles what they think Nue is.

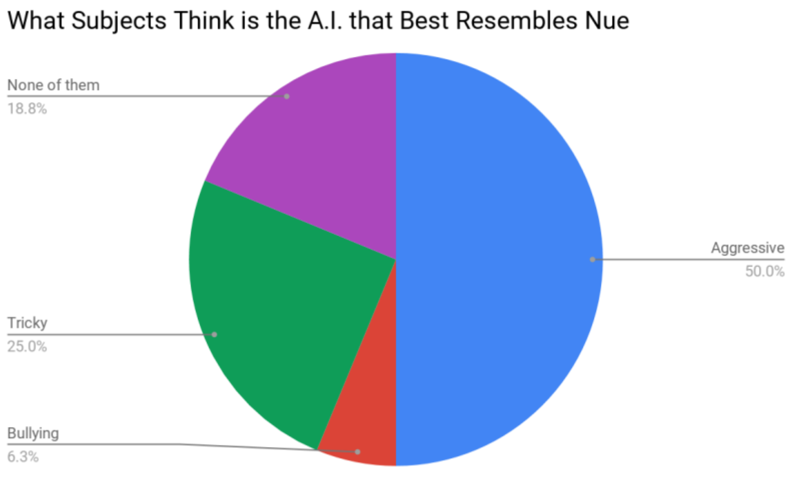


Figure 77: Subject's opinion on what is the A.I. that best resembles Nue

Therefore Aggressive is the most believable, followed by Tricky, Bullying and Cowardly are not believable at all.

6.7.1 Visuals

Out of 7 subjects who know what Nue looks like, 3 of them think our Nue in the game visually resembles what they know about it, 4 of them think our Nue in the game does not visually resemble what they know about it.

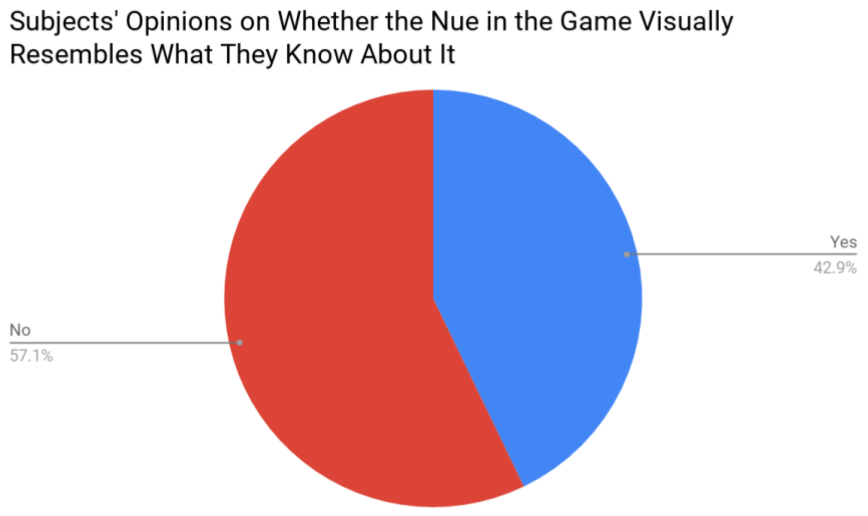


Figure 78: Subject’s opinion on whether the Nue in the game visually resembles what they know about it

Chapter 7: Conclusion

From the analysis, we can see that Aggressive A.I. is very believable and moderately recognizable; Tricky A.I. is moderately believable and moderately recognizable; Cowardly A.I. is not at all believable but very recognizable; Bullying A.I. is not very believable and not very recognizable.

The reason for Aggressive A.I.'s success is largely due to its intuitive naming, fitting well with the portrayed Nue character and matching people's expectation of what an NPC in ARPG should be like.

The reason for Bullying A.I.'s failure is due to its confusing definition, preventing people from actually knowing that it is bullying. It is also too similar to Aggressive A.I., even a bit weaker due to its longer delay for melee and cooldown for charge.

The cowardly personality succeeds on recognizability by not being aggressive, but it is also completely detached from the Nue character and people's expectation, therefore failed to convince anyone that it is what Nue should be like.

The tricky personality succeeds on meeting people's expectation about being aggressive, while not meeting people's expectation about its behavior by charging with shorter delay and only charge when it has a high chance of hitting, giving people a surprise factor. It is also the only one that actively responds to player's input, making it very responsive.

We cannot draw any conclusion about which A.I. algorithm is better, as we did not make two A.I.s with different algorithms but same personality to compare with.

However, the testing methodologies have issues that may affect the accuracy of the result.

We can tell what a person's personality is not because we know how to do it naturally, but through comparing others, and creating a standard of which personality is which in our mind. A.I.s in our game are different, they need to attack in order to proceed the game, therefore will and must contain certain level of aggression. Because our testing asked for subjects to answer the survey after each round, they couldn't compare the first round to anything, so most of them just guessed. For the second round the only reference for them was the first round, and since the second A.I. (Bullying) plays less aggressively, most of them just picked Tricky or Cowardly, because it is, in a sense, more cowardly than the first one. For most subjects, they only start to understand what the standard is when playing the third A.I. (Cowardly). This is proved by one of the responses for the third A.I.: "Despite my previous response, this is the cowardly Nue. He was cowardly the entire time and would try to walk away from me as I approached him", as the subject thought the second A.I. is cowardly because he hadn't played the third A.I. yet. Therefore it is better to give the survey to subjects after they played through all 4 rounds, so they can compare between all of them before answering questions, and recognizability of each A.I. should be a lot higher, especially for the first A.I. (Aggressive) and the second (Bullying).

The wording of the convincing rating survey question is not very clear, and three subjects misunderstood the question as to rate how well each A.I. is. It is even confusing for us, the creator of the survey, when not paying attention. It should be worded like "on a scale of 1 to 5, rate how confident you are of your answer to the previous question".

The definition of bullying was not explained well enough to the test subjects. Most of them just avoided the term, because they weren't sure what it means exactly. Aggressive, tricky and cowardly are very simple and straightforward terms, but bullying is more complicated, as it inherits both aggressiveness and cowardness. People's objective understanding of bullying varies greatly, and this prevents them from confidently picking bullying.

Bullying A.I. is not very recognizable; therefore it is never going to be believable, as subjects won't pick it as the best one if they don't know that the second A.I. is supposed to represent bullying. We should tell subjects what personality each A.I. represents after they play through all of them, so they will at least know what they are.

Finally, many subjects failed to complete every round; they died before they can see the changes of behavior for some A.I.s over time. For example, Bullying A.I. will run away when its health is lower than the player’s and will be aggressive again when it regains health advantage. This behavior is only seen by 1 subject as he/she wrote it on the explanation.

Chapter 8: Post-Mortem

In this chapter, we assessed the experience from our game and AI development. We focused on the knowledge we gained from the experience, things that went well for us, things that went wrong for us and future development we have for the our game.

8.1 What We Learned?

Throughout the process, we gained many unforgettable experiences. The most important thing we learned was that communication is very important to having a productive team work. Being in a team means being on the same page and that means talking to each other as much as possible, building rapport. In addition, being a team player and an understanding member of the team can go a long way in forming a great team dynamic. We learned firsthand what a poor team dynamic can lead to. That includes not knowing what each other wrote, not knowing who is doing what, and not knowing everyone status both personally and work related. Due to the dissonance in our team, we faced difficulties getting work done and of course speaking to each other on equal terms. Eventually we turned it around by interacting more and cooperating more with each other, but we definitely lost times while facing communication issues.

The second thing we learned was how to use Unity and C#. Going into this project, only one member of the team has experience making a game while the rest had none. Learning how to code in C# was not a problem for the team because the learning curve was not difficult, but learning to use Unity Engine took time.

The third thing we learned was the game development process. We were all familiar with the software development cycle from our own experience in either job and/or software engineering course.

The fourth thing we learned was actually making the game. We started from scratch and went through the game development cycle to create a game. That includes integrating animations with controllers and creating game mechanics like dodging, attacking, charging, etc. We also learned how box colliders work and the possibilities we could do with it. We also learned how to make a cutscene using a combination of unity animation plugins and through code. It was the basics in creating a game, but they were very valuable information.

The final thing we learned was developing AI. Coming into this project, we had no substantial knowledge of AI development. We were interested in AI and it was the main component of our game. During the course of the project we learned about many AI architectures like behavior tree, utility-based, state machine and machine learning. Although we used a framework for behavior tree, Panda BT, we were still able to learn how it works and it effectiveness by creating the behavior tree ourselves to find the right combination of behaviors. As for the utility-based, we were also using a framework but the real work was developing properties and finding the right functions for each property in order to define each action that the AI can do. Overall, we have learned so much about AI development from this project experience.

8.2 What Went Right?

The first thing is that we have a certain style of art. The team is formed by three Computer Science major students but we designed a suitable art style that goes well with our game in the game design phase. It was a wise decision we made to go with 2D art style because it was easier to make than 3D art and we had a pretty satisfying result in the end. The art for background and characters were done by two different people, yet the art style match with each other which is a good thing as well.

Game mechanics were well polished during the game development phase. We fixed many small issues and bugs and added some new features in the end of game development phase, such as special effects and tutorial session.

We successfully rescaled our goal of making an entire game about yokai and scaled it down to different personalities of one Yokai. All the personalities were finished without bugs and has their unique personalities. We also tried different ways, including utility AI, state machine and behavior tree to create these personalities as well.

Most of our test results were positive and some of the players actually enjoyed the gameplay pretty much. We all think the results are good enough as evidence in a solid report.

8.3 What Went Wrong?

This project went through some horrible team communication and collaborative work issues.

We relied too much on a modular game's architecture in order to work with machine learning. We spent too much time on that and no one were really clear about how to implement that in our game. After the architecture is finally suitable for machine learning, the ML-Agents (v0.4 at the time) machine learning library ended up being buggy, and had to be given up entirely. A lot of time is wasted, and all of us lost a lot of motivation.

ML-Agents works as intended when it calculates every frame, but the order of function call is messed up when using on demand decision making. Normally, for each decision cycle, it will observe the environment, make the decision, and take the action. After it calculates for a certain number of steps, it will reset the environment, and goes through the process of resetting the academy (and the environment), resetting the agent, and start decision cycle again. All the calculations are handled by external Python scripts, and the environment parameters are synced between Unity and the scripts. However, for ML-Agents v0.4's on demand decision making, its Unity part collects observation when it should be resetting the academy, and thus breaking the sync between it and the external scripts. This prevented us from performing any training over 1000 steps, as well as not being able to reset the environment.

We overestimated our ability and set the goal too high. We were going to do adaptive A.I. with multiple algorithms, a game with progressive abilities, a full storyline, multiple Yokais and background stories for each of them. We are a team with only Computer Science majors, and two of us are completely new at making games. We had to make a complete game and do A.I. research at the same time, it is simply not possible to make things too complicated. We aimed too high and fell short; this is the other reason why we lost our interest.

8.4 Future Development

Initially, our plan for the game was to develop a full game. Our initial plan consists of having the ability to choose the character gender, having BGM, more battle mechanics, more enemies and a progression system. For the future development, we want to complete the game that we planned out in the beginning.

Currently the main character only has two combat abilities, melee and dodging. We want to add another option for combat, a range attack using the idea of onmyoji. This allows the players to have more variations in their play style.

Many other things to add in the game like more interactions with the environment such as finding items, buffing one's abilities, and learning about the stories of Yokais. There would be a progression system that allows the player to adjust their abilities as they play through the game.

In addition, we want to add more enemies to our game. There are abundant of interesting Yokai in Japanese Mythology so we want to incorporate their stories in our game. We want to make the AI adaptive by having the enemy learns from the player and make it tougher for the player. We also want to improve and expand on the enemy's personality by giving them backstories and more interaction with the player other than combat.

References

[1] Access. Kodai-Ji Temple, Summer 2018

[2] Alicia Joy. 2016. “Japan's Spookiest Urban Legends And Myths.” Culture Trip, 23 Aug. 2016, theculturetrip.com/asia/japan/articles/best-japanese-urban-legends-and-myths/.

[3] Anime-Planet Community. Best Youkai Anime. Retrieved October 15, 2018 from https://www.anime-planet.com/anime/tags/youkai

[4] Anon. 2016. Shintoism, Animism; Kami, Yokai. (December 2016). Retrieved October 14, 2018 from https://japaneseyokai.wordpress.com/2016/12/05/shintoism-animism-kami-yokai/

[5] Anon. 2018. C Sharp (programming language). (October 2018). Retrieved October 15, 2018 from https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[6] Anon. 2018. Different Game Design Styles: Japan vs The West. (May 2018). Retrieved September 11, 2018 from https://www.gamedesigning.org/gaming/game-design-styles-japan-vs-west/

[7] Anon. Circular Diagram - Big Five Personality Dimensions. Retrieved October 15, 2018 from https://www.edrawsoft.com/personality-circular-diagram.php

[8] Anon. Mononoke. Retrieved September 13, 2018 from https://tvtropes.org/pmwiki/pmwiki.php/Anime/Mononoke

[9] Anon. The Ultimate Yōkai Guide. Retrieved September 13, 2018 from https://www.wattpad.com/393248311-the-ultimate-yōkai-guide-nue

[10] Anon. Wizard of Legend - Magical Spell Slinging Combat. Retrieved October 15, 2018 from http://wizardoflegend.com/

[11] Anon. ホーム - 高台寺. Retrieved October 15, 2018 from http://www.kodaiji.com/

[12] Anon. 昆布が美味い. Retrieved October 15, 2018 from https://blog.goo.ne.jp/travellgoo/e/28cfa87c857ccc6e0bc009be40b04719

[13] Berit Larsen. 2014. What characterises people who stay calm in crisis situations? (December 2014). Retrieved October 12, 2018 from https://psychology.stackexchange.com/questions/8939/what-characterises-people-who-stay-calm-in-crisis-situations

[14] Brett. 2018. How To Win a Street Fight. (May 2018). Retrieved October 12, 2018 from https://www.artofmanliness.com/articles/how-to-win-a-street-fight-in-7-simple-steps/

[15] Bill Merrill. 2014. Building Utility Decisions into Your Existing Behavior Tree. In Game AI Pro. CRC Press/Taylor and Francis Group, 127–136.

[16] Binding of Isaac: Rebirth Wiki. 2018.(July 2018). Retrieved October 15, 2018 from https://bindingofisaacrebirth.gamepedia.com/Binding_of_Isaac:_Rebirth_Wiki

[17] Blizzard Entertainment, 1996, Diablo (series) on platform: Microsoft windows, Classic Mac OS, PlayStation, Xbox 360, Nintendo Switch

[18] Chattyfeet, Shovova, and Yugen Tribe. 2018. What is Kawaii? Discover What Led to Japan's Culture of Cuteness. (August 2018). Retrieved October 13, 2018 from https://mymodernmet.com/kawaii-art-japanese-culture/

[19] Daniel Hilbum. 2014. Simulating Behavior Trees. In Game AI Pro. CRC Press/Taylor and Francis Group, 99–111.

[20] David J. Llewellyn and Kerry M. Wilson. 2003. The controversial role of personality traits in entrepreneurial psychology. Education Training45, 6 (2003), 341–345. DOI:http://dx.doi.org/10.1108/00400910310495996

[21] Douglas Schules. 2015. Kawaii Japan: Defining JRPGs through the Cultural Media Mix. (December 2015). Retrieved September 13, 2018 from https://www.kinephanos.ca/2015/kawaii-japan/

[22] Erin Rushing. 2016. Toriyama Sekien's Spooky Parade. (October 2016). Retrieved October 13,

	2018 from https://blog.library.si.edu/blog/2016/10/14/toriyama-sekiens-spooky-parade/
[23]	George Simon. 2011. Understanding the Aggressive Personalities. (March 2011). Retrieved October 13, 2018 from https://counsellingresource.com/features/2008/11/03/aggressive-personalities/
[24]	James T. Tedeschi. 2003. The Social Psychology of Aggression and Violence. In: Heitmeyer W., Hagan J. (eds) International Handbook of Violence Research. Springer, Dordrecht (2003)
[25]	Jeffrey Georgeson and Christopher Child. NPCS AS PEOPLE, TOO: THE EXTREME AI PERSONALITY ENGINE . https://arxiv.org/ftp/arxiv/papers/1609/1609.04879.pdf
[26]	Jeremy A.GlasserLeen-Kiat Soh. 2004. AI in Computer Games: From the Player's Goal to AI's Role. (2004). Retrieved October 13, 2018 from http://digitalcommons.unl.edu/csetechreports/86
[27]	Kansas Safe Schools Resource Center.Retrieved October 15, 2018 from https://community.ksde.org/Default.aspx?tabid=3913
[28]	Lottie Wilson. 2017. The rise of open-world gaming. (September 2017). Retrieved October 15, 2018 from https://www.gamasutra.com/blogs/LottieWilson/20170927/306482/The_rise_of_openworld_gaming.php
[29]	Magalie Ochs, Nicolas Sabouret, and Vincent Corruble. Simulation of the Dynamics of Non-Player Characters' Emotions and Social Relations in Games.
[30]	Matthew Meyer. Yokai.com. Retrieved September 13, 2018 from http://yokai.com/nue/
[31]	Matthew W.G. Dye, C.Shawn Green, and Daphne Bavelier. 2009.(2009). Retrieved October 13, 2018 from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2871325/
[32]	MichaelHouse. 2013. Difference between Decision Trees & Behavior Trees for Game AI. (March 2013). Retrieved October 16, 2018 from https://gamedev.stackexchange.com/questions/51693/difference-between-decision-trees-behavior-trees-for-game-ai
[33]	Michael Rougeau. 2018. The Binding of Isaac: Rebirth review. (March 2018). Retrieved October 15, 2018 from https://www.digitaltrends.com/game-reviews/binding-isaac-rebirth-review/
[34]	Mike Lewis and Kevin Dill. 2015. Game AI Appreciation, Revisited. Game AI Pro 2(2015), 3–18. DOI:http://dx.doi.org/10.1201/b18373-3
[35]	Murray R. Barrick and Michael K. Mount. 1991. The Big Five Personality Dimensions And Job Performance: A Meta-Analysis. Personnel Psychology44, 1 (1991), 1–26. DOI:http://dx.doi.org/10.1111/j.1744-6570.1991.tb00688.x
[36]	Paul Ratner. 2016. Why Do Japanese People Love Cuteness? Learn the Science of "Kawaii". (May 2016). Retrieved September 13, 2018 from https://bigthink.com/paul-ratner/why-do-the-japanese-love-cute-things
[37]	Penelope Sweetser, Daniel Johnson, Jane Sweetser, and Janet Wiles. 2003. Creating engaging artificial characters for games. In Proceedings of the second international conference on Entertainment computing (ICEC '03). Carnegie Mellon University, Pittsburgh, PA, USA, 1-8.
[38]	Richard W. Wrangham. 2018. Two types of aggression in human evolution. (January 2018). Retrieved October 11, 2018 from http://www.pnas.org/content/115/2/245#xref-ref-62-1
[39]	Savvytokyotop. 2018. Setsubun: Japan's Bean-Throwing Tradition. (February 2018). Retrieved October 13, 2018 from https://savvytokyo.com/fun-family-activities-for-the-bean-throwing-holiday-of-setsubun/
[40]	Simon Norton. 2018. Yokai: the monsters and demons of Japanese folklore. (July 2018). Retrieved October 13, 2018 from http://www.4corners7seas.com/japan-folklore-yokai-tengu-kirin-kappa-kitsune/
[41]	Steve Rabin. 2014. Game AI pro: Collected Wisdom of Game AI Professionals. CRC Press/Taylor and Francis Group, 2014.

[42]	Steve Rabin. 2017. Game AI Pro 3: Collected Wisdom of Game AI Professionals. CRC Press, Taylor & Francis Group, 2017.
[43]	Tanya X. Short. 2017. Designing Stronger AI Personalities. (September 2017). https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15889
[44]	Umarov, I., & Mozgovoy, M. (n.d.). Creating Believable and Effective AI Agents for Games and Simulations. Advances in Systems Analysis, Software Engineering, and High Performance Computing Contemporary Advancements in Information Technology Development in Dynamic Environments,33-57. doi:10.4018/978-1-4666-6252-0.ch003
[45]	Unity. Multiplatform. Retrieved October 15, 2018 from https://unity3d.com/unity/features/multiplatform
[46]	Wiki. 2018.(May 2018). Retrieved October 15, 2018 from https://wizardoflegend.gamepedia.com/Wizard_of_Legend_Wiki
[47]	Wikipedia. 2018. Bullying. (October 2018). Retrieved October 15, 2018 from https://en.wikipedia.org/wiki/Bullying
[48]	Wikipedia. 2018. Nue. (September 2018). Retrieved October 15, 2018 from https://en.wikipedia.org/wiki/Nue
[49]	Wikipedia. 2018. The Great Wave off Kanagawa. (October 2018). Retrieved October 15, 2018 from https://en.wikipedia.org/wiki/The_Great_Wave_off_Kanagawa
[50]	Yamagishi Hayase. 2015. The Spectacular World of the Yokai. 公益財団法人国際文化フォーラム, Aug. 2015, www.tjf.or.jp/clicknippon/en/mywayyourway/07/post-17.php.
[51]	Yoann Bourse. 2012. Artificial intelligence in the Sims series, (2012, February)http://yoannbourse.com/ressources/docs/ens/sims-slides.pdf
[52]	Zack Davisson. 2014. A Brief History of Yokai. (January 2014). Retrieved October 13, 2018 from https://hyakumonogatari.com/2013/02/05/a-brief-history-of-yokai/

Appendix

Raw Survey Response

Pre-test Survey

ID	Timestamp	Are you from Japan	Do you know what the Yokai, Nue (鵺), look like ?	Do you know what Nue is known for ?	What is Nue known for ?	Where did you learn about Nue ?
1	9/24/2018 14:03:49	No	Yes	No		
2	9/24/2018 14:49:35	No	No	No		
3	9/25/2018 14:30:44	No	No	No		
4	9/25/2018 15:20:46	No	No	No		
5	9/25/2018 16:06:30	No	Yes	No		
6	9/25/2018 17:25:14	Yes	No	No		
7	9/26/2018 15:52:31	No	Yes	Yes	monkey	animation
8	9/26/2018 16:11:47	No	No	No		
9	9/26/2018 16:39:33	No	No	No		
10	9/26/2018 17:07:37	Yes	No	No		
11	9/26/2018 17:36:29	Yes	Yes	Yes	youkai, snake and tiger	anime
12	9/27/2018 13:44:01	No	Yes	No		
13	9/27/2018 13:45:12	No	No	No		
14	9/27/2018 13:59:19	No	Yes	No		
15	9/27/2018 14:00:19	No	Yes	No		

16	9/30/2018 14:07:57	No	No	No		

Round 1

ID	What do you think the personality of this Nue is?	On a scale from 1 - 5, rate how convincing the Nue is at representing the personality you chose in the previous question?	Explain your reasoning from previous two questions.
1	Aggressive	5	Nue was attacking and trying to kill me
2	Bullying	5	I can't pass the round 1.
3	Aggressive	4	My main reasoning for the previous answer is that the Nue kept attacking me constantly and was always chasing me. He never stopped chasing.
4	Aggressive	3	It's tried to kill me
5	Aggressive	5	かわいです
6	Tricky/Clever	5	鵺がプレイヤーの方向を向くのが敏感だったから
7	Aggressive	2	rest for a long time
8	Cowardly	4	he stays there and doesn't move for a long time
9	Aggressive	4	it would suddenly attack me
10	strong	5	he always fight with me face to face
11	Aggressive	4	Its attacks looks strong
12	Aggressive	2	It would follow and then charge to attack, and this fit more than the other personalities
13	Aggressive	4	quick movements towards the player

14	Aggressive	5	He is big and attacks you
15	Tricky/Clever	3	Would stay still then attack when I go closer
16	Aggressive	4	He just kept attacking me. But there were some pauses

Round 2

ID	What do you think the personality of this Nue is?	On a scale from 1 - 5, rate how convincing the Nue is at representing the personality you chose in the previous question?	Explain your reasoning from previous two questions.
1	Aggressive	3	Nue was attacking but the first round was more aggressive
2	Aggressive	4	I can pass!
3	Cowardly	5	By the end, this nue kept running into the wall or corner. I thought it was bugged, but it must just be scared.
4	Tricky/Clever	3	It's waiting at the corner and hit
5	Tricky/Clever	5	かこいです
6	Aggressive	3	鵺がプレイヤーに向かってよく走ってきたから
7	Cowardly	3	not aggressive
8	Aggressive	3	it attacks you only when you are in a certain range
9	Tricky/Clever	5	it moves very fast
10	Aggressive	4	he will approach me and dash at me
11	Bullying	4	this round it runs away after attack me

12	Bullying	3	It tended to follow more closely and attack more often
13	Tricky/Clever	3	attack pattern makes you think they have given up and then they attack you
14	Aggressive	4	He always takes the opportunity to attack you when he is close by
15	Aggressive	5	runs at you
16	Aggressive	4	I cannot see when it is attacking me, but it is aggresive

Round 3

ID	What do you think the personality of this Nue is?	On a scale from 1 - 5, rate how convincing the Nue is at representing the personality you chose in the previous question?	Explain your reasoning from previous two questions.
1	Cowardly	5	Nue is not attacking, he just attack to defend himself. Actually it seems like he tries to be friendly but he is shy because he tries to stay close but when you approach him, he run away.
2	Cowardly	5	I don't need to eat health.
3	Cowardly	5	Despite my previous response, this is the cowardly Nue. He was cowardly the entire time and would try to walk away from me as I approached him.
4	easier than before	1	just stay
5	Cowardly	1	まだです
6	Cowardly	1	プレイヤーに向かって走ってきたりせずに、適当に鵺がダッシュしてたから
7	Cowardly	5	its not attacking me at all

8	Cowardly	4	he doesn't attack you if you don't attack him
9	stupid	4	he doesn't even chase me after being attacked by me
10	Cowardly	4	Basically it faces backwards
11	not clever	4	It attack other way
12	Cowardly	5	It ran away as you moved towards it and used the charge attack to flee
13	Cowardly	5	monster is always running away from the player
14	Cowardly	5	He runs away and only attacks when needed.
15	Cowardly	4	keeps running
16	Aggressive	4	I am better at the game now. so the Nue is not as aggressive as I thought. Also the other options like bullying are not describing the behaviour of the nue

Round 4

ID	What do you think the personality of this Nue is?	On a scale from 1 - 5, rate how convincing the Nue is at representing the personality you chose in the previous question?	Explain your reasoning from previous two questions.
1	Tricky	5	You dont know when exactly he is going to attack you. A little bit tricky. It was the most difficult to defeat.
2	Aggressive	5	I do not know
3	Bullying	4	This boss was an asshole. He kept charging his heavy damaging move, then not using it until I was too close to possibly dodge it.
4	Aggressive	4	first half it very aggressive

5	Tricky/Clever	5	早いです
6	Tricky/Clever	5	鵺がよくカウンタをしてきて、動きがすごく敏感にプレイヤーに接近してきたから
7	Tricky/Clever	4	it reacts to my actions
8	Tricky/Clever	3	he sometimes pretend to run away from me, but once I get close to him, he will bump at me
9	Aggressive	5	he moves when I move, and I cannot predict when he is going to attack me.
10	Tricky/Clever	4	He will not come if you are away, but as you get closer he will attack you.
11	Tricky/Clever	5	It find out whether I move or stop.
12	Tricky/Clever	4	It would lure the player in and then charge attack them
13	Tricky/Clever	5	monster runs away to draw the player in and then turns to attack
14	Tricky/Clever	5	Runs away then when you get close he will turn and attack you or rush onto you.
15	Aggressive	4	keep running's アンドｃあｎ：と dodge
16	Aggressive	5	The Nue was attacking relentlessly on this level

Post-test Survey

Do you feel that our Nue visually resembles what you know about it from somewhere ?	Based on your knowledge, which of the personality listed best resembles Nue ?	Is there anything else you would like to comment about your experience, or do you have suggestions for us?	You chose "None of the above", please explain why so.
It has a look of what I have seen before	None of the Above	i want more blood	Because I dont have knowledge about Nue's personality
Yes	Aggressive	I think when I enter round 1, I didn't know that I need stamina to attack. A better tutorial is needed.	
It reminds me a bit of how Sun Wukong looks in some depictions.	Tricky	Make stamina refill more quickly and add music.	
no	Aggressive	BGM	
いいえ	Aggressive	がんばつて	
鵺をよく知らないので、完全に似ているとは言えないが、このゲームの鵺のキメラっぽい雰囲気はよく出ていたので、おそらくだいたい似ていると思う	Aggressive	鵺を検索してみて、画像を見たら鵺がライオンに近い感じだった。しかしこのゲームの鵺はどちらかというと毛深い猿の感じがしたので鵺は四足歩行の方がいいのではないか バトル開始時にすぐに始まってしまうので、少しカウントを入れてからバトル開始の方がいいと思った	
no	Tricky	none	
I don't know it	None of the Above	I don't even know this monster, I can't answer the above two questions	I don't know what it originally looks like
I don't know what is it	None of the Above	it feels like they all have different personalities. sometimes its stupid, sometimes its aggressive.	I don't know this monster at all.

I don't know it.	Aggressive	too strong	
yes	Bullying	its a nice game, Difficulty is appropriate	
No, but the last time I heard about Nue was awhile ago.	Aggressive	Try and always keep the Nue in the view of the player	
no	Tricky	the attack patterns are super easy, and the monster will not move if you don't in the 4th fight	
Yes	Aggressive	Very interesting concept, hard to distinguish between 1 and 2 nues	
no	Tricky	@「えええ	
I never knew about Nue	Aggressive	I would like to see the Nue attacking me. The seems to be a lag or something.	

Joint Project with Ritsumeikan Students

We had a joint project with Ritsumeikan Students from our Lab. The project was to create a puzzle solving robot using a makerplot. The project had four phases. the first phase was being able to scan a puzzle. The second phase was being able to solve the puzzle. The third phase was having the program be human supported. The last phase was an full-on automatic solving robot.

The problem we ran into during the project was that it happened during the summer vacation for the Ritsumeikan Students. Many students either went home or completely forgot about the project. In addition, there was a huge language barrier in the group. The Ritsumeikan Students could only speak Japanese while we could only speak English. We knew they had the ability to read English, but they never tried to speak with in English. The lack of communication made it very difficult to move the project forward.

In the end we could only go as far as phase one. Our robot can move and it can scan puzzle pieces but that was all that it could do.