The Impact of Indirect Advantages on Artificial Intelligence Agents' Behaviour in Video Games

Master Thesis

Sebastian Martin Young

Aalborg University
Medialogy MSc.

Medialogy MSc.
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
The Impact of Indirect Advantages on Artificial Intelligence Agents' Behaviour in Video Games

Theme:
Artificial Intelligence in Games

Project Period:
Spring Semester 2022

Participant(s):
Sebastian Martin Young

Supervisor(s):
Georgios Palamas

Copies: 1

Page Numbers: 61

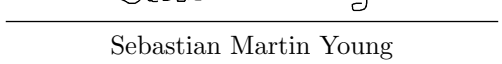
Date of Completion:
May 25, 2022

Abstract:

The purpose of this paper was to explore and observe whether artificial intelligence agents outfitted with indirect advantages and communication methods could develop new and interesting behavioural patterns compared to traditional State of the Art Ad-Hoc design methodologies. The hypothesis of the paper was that agents when provided with indirect cooperative advantages would behave less predictably and in a more diverse manner. A game prototype was developed in Unity's game engine, and two agents types with an adversarial relationship were designed to play through the game: Monster Agents and Friendly NPC Agents. The Friendly NPC Agent was further divided into three groups: Finite State Model, Reinforcement Learning: Perception Bonus Advantage, and Reinforcement Learning: Pack Tactics. The two Reinforcement Learning NPC were trained against the Monster Agents, and finally each Friendly NPC Agent ran for a session of 12 hours to collect data on their positions and game score. The results from the score test show that the Ad-Hoc Finite State Machine model significantly outperforms the Reinforcement Learning models in terms of completing the games objectives. The results from the Heatmap Test displays a chaotic and unpredictable behaviour from the Reinforcement Learning models. However, due to the limited training period of the agents, it is not possible to measure the exact impact the indirect advantages have on the model, and the alternative hypothesis can thus not be accepted.

Contents

Aalborg University, May 25, 2022



Sebastian Martin Young
syoung17@student.aau.dk

1 Introduction

The use of artificial intelligence in game design has been growing in recent years. AI agents have been used to create new and interesting gameplay experiences in a variety of genres. In this paper, we investigate the impact of indirect advantages on AI agents' behaviour. Indirect advantages are those that are not directly related to the agent's task or goals, but may nevertheless be beneficial to the agent. The aim is to explore and observing whether artificial intelligence agents outfitted with indirect advantages and communication methods could develop new and interesting behavioural patterns compared to traditional State of the Art Ad-Hoc design methodologies.

2 Analysis

The video game market has over the last few years exploded in revenue, growing larger than the movie and music industries combined, and encompasses a broad spectrum of different genres. Games can be simplified into player(s) versus player(s) also known as *PvP* or player(s) versus game/system also known as *PvE*, with a variety of sub-categorisations under each such as cooperation, competition, team competition etc[Fullerton 2018]. This paper focuses on designing one of the commonly recurring elements of the *PvE* genre, the artificial intelligence of Non-Player Characters(NPCs), with the goal of eliminating one of their core weaknesses, the predictability of their behaviour[Lara-Cabrera et al. 2015]. To that end, this analysis will explore different areas of how to design Non-Player Characters, potential solutions, and game design as a whole to create an environment to implement and test the proposed solution.

2.1 Artificial Intelligence in games

Artificial Intelligence can be found in various elements of game design, and there is no optimal design solution that necessarily fits all design spaces. This section will cover some of the most commonly used AI design systems utilized in games, and their pros and cons. As there is a great deal of freedom of how individual design algorithms are created, this design exploration will try to generalize their uses. State of the Art 2.3 will cover a detailed exploration into a selection of different artificial intelligence design systems used by State of the Art games.

2.1.1 Ad-Hoc Behaviour Authoring

The area of Ad-Hoc Behaviour Authoring methods cover a wide range of the most commonly used AI systems for NPCs in games such as *Finite State Machines*, *Behaviour Trees* and *Utility-Based AI* models.

Finite State Machines (FSM)
Finite State Machines (FSM), defined by a set of finite set of *states* that the AI can be in, *transitions* between the various states based on a number of parameters, and a set possible *actions* available based on the current state[Yannakakis and Togelius 2018a]. FSMs are versatile as each state, transition and action can be coded and tailored by a programmer to the individual goal of the agents, across nearly any type of game and predictable situation that the AI can find itself in. However, they can become incredibly complex to design on a large scale, as the design space the AI finds itself in expands, the more the FSM needs to be iterated on to account for

new features. While FSM agents tend to be robust in their predictability, this also results in a lack of flexibility to adapt to changes and act dynamically[Yannakakis and Togelius 2018a].

Behavior Trees
Behavior Trees, closely related to *FSM*, are able to transition between a set of finite tasks / behaviours[Yannakakis and Togelius 2018b]. This allows well designed behaviour trees to be able to perform complex behaviours based on few simple tasks/goals, and has been a stable AI design for a large number of games. They provide for very flexible design, quick and easy testing and debugging. However, similar to FSM, they lack adaptability and their behavioural states easily become predictable[Yannakakis and Togelius 2018b].

Utility-Based AI
Utility-Based AI is a design focused around a design-making process based on a utility depending on the situation[Yannakakis and Togelius 2018c]. The AI explores a set variety of options depending on its current state / behaviour, and using a utility function, weighs the best course of action on the behavioural transition based on a variety of relevant parameters in that specific situation[Yannakakis and Togelius 2018c]. Noise is often added to such a utility function, to add a bit of randomness to their decision-making thus lowering the predictability of a given action the Utility AI will take[Yannakakis and Togelius 2018c].

2.1.2 Artificial Neural Networks

Artificial Neural Networks(ANNs) originally designed to model the way a biological brain processes information, operates, learns and performs various tasks[Yannakakis and Togelius 2018d]. The model contains a set of interconnected processing units called perceptrons which contain both a weight **w** and a bias **b**. These perceptrons can be connected in a multitude of ways in the hidden layer between the initial input layer and the output layer. Neurons are fed to an activation function such as a *Binary Step Function*, *Linear Activation Function* or or *Sigmoid/Logistic Activation Function*, to just name some of the more commonly used. These activation functions produce an output for the neuron based on its weight, bias, and the type of activation function, which can be fed further into the next layer.
Due to their various forms and the flexibility in design models, *Artificial Neural Networks* can be used for a variety of areas such as pattern recognition, robot and agent control, game-playing, decision making, etc[Yannakakis and Togelius 2018d]. ANNs create their own rules on how the relationship between the input layer and the output layer, and are thus able to function when presented with unknown input data, if they do not differ too much from the trained data.

2.1.3 Genetic Algorithms

Genetic Algorithms (GAs) are based on mimicking evolutionary biology techniques. With a wide range of usage, *GA* is in essence a search algorithm that attempts to find a possible solution, not necessarily the best solution. *GAs* are modelled over the genetic evolutionary aspects of passing on genes from one generation to the next referred to as the *Crossover*, along with *Mutation* of the genes to generate new possible gene-makeups. Each member of the population of the initial generation Gen_I starts off with X random genes, this could be an order of operations such as moves: *Up, Down, Left, Right* in the example of finding a path through a maze. A fitness function determines how *fit* an individual is, and assigns each member of the population its own individual fitness score, which represents the probability that an individual

will be selected for reproduction. At the next stage, the fittest individuals of the population are naturally selected to pass on their genes to the next generation. This process repeats itself until the new generation Gen_n has the same population size as the previous generation Gen_{n-1}. *Genetic Algorithms* are very versatile in their usages, with a great freedom in gene design, choice of *crossover* and *mutation* rates and population sizes and generations. *GAs* can be incredibly useful tools, due to their speed, and use of solution approximations over a great range of problems. However, they risk falling into a local maximum due to their commonly designed *Elitism* that always picks the best candidates to put into the next generation.

2.1.4 Reinforcement Learning

Reinforcement Learning inspired by behaviourist psychology is a machine learning approach that is based on the idea of positive and negative feedback in an environment, similar to how humans and animals learn to make decisions[Yannakakis and Togelius 2018e]. An agent explores the environment and collects a set number of observations which are used to determine its current ***State*** *(s)* at each time-step. The agent then decides to take an ***action*** *(a)* from currently available actions. In response the environment can provide an immediate ***reward*** *(r)* r[Yannakakis and Togelius 2018e]. Over time, the agent begins to learn the relationship between its ***States***, ***Actions*** and its possible ***Rewards***, and then makes decisions or ***Action-State Pairs*** to maximize its potential rewards[Yannakakis and Togelius 2018e]. Agents are also able to learn long-term non-immediate rewards given sufficient training and understanding of their environment. *Reinforcement Learning* has broad usage within a wide range of fields and can be used to solve complex problems and adapt to a variety of changes in its input-space when properly trained. As agents' goal is to learn the relationship between *Action-Space Pairs* and both the positive and negative *Reward Signals*, this is done by, incorporating randomness in their environment or generating the environment procedurally , which can then help agents better understand these relationships and allow the agents to complete their goals and subgoals in new territories.

Policies - PPO, SAC & MA-POCA
Three different training policies were examined and considered for the training policies of this project's agents. They are, ***Proximal Policy Optimisation (PPO)***, ***Soft Actor-Critic (SAC)***, and ***MultiAgent Posthumous Credit Assignment (MA-POCA)***.

Proximal Policy Optimisation (PPO) - On-Policy
PPO uses an on-policy learning method, in that the model updates the policy by evaluating the state-action pairs of previous versions of itself [Sutton and Barto 2018]. This results in future iterations of the model having gradually less and less random actions as the system attempts to achieve rewards from specific state-action pairs that it has already discovered [OpenAI 2020]. As new actions are evaluated, based on previous state-action pairs, the model can not stray too far away from its defined policy, as this can result in it becoming stuck in the local optima, which limits the ability of new action pairs to be explored. Unless the policy is provided with a new incentive to explore new state-action pairs, the global optima can not be achieved.

Soft Actor-Critic (SAC) - Off-Policy
Off-Policy methods such as SAC, contain two different policies, a *target policy* and a *behaviour policy*. The *target policy* is the policy that becomes the optimal policy by evaluating the different state-action pairs produced by the *behaviour policy*, which is more exploratory and does not necessarily get impacted by the *target policy*. This means that the *behaviour policy* can be more greedy in its attempts to find the global optima [Sutton and Barto 2018].

MultiAgent Posthumous Credit Assignment (MA-POCA) - Centralized Critic
MA-POCA is a cooperative learning policy that allows agents to both receive individual rewards and rewards as a group. In addition, should agents' episode early, they will still be provided with information as to whether their actions contributed in a positive way for their group.

Curriculum Learning
Curriculum learning helps to guide an agent by providing changes to the environment over time. This aim is to ensure that the model trained, is always optimally challenged and does not become too easy over time, or start out too hard. An environment is created which makes it easier for the agents to receive immediate extrinsic rewards from the environment. This helps reduce the amount of exploration the agent needs to do. Changes are gradually made to the environment, to make it more similar to the intended environment that the agent is trying to learn its objective for. When these changes occur, they are generally dependent on either the average cumulative reward the agent has received over a range of runs, or how long the model has been training.
Curriculum Learning's methodology can assist in training agents in their understanding of how to complete sub-goals. However, a balance between the rewards awarded and the criteria for next lesson has to be found

2.1.5 Modelling behaviour & Intelligence

With such a wide selection of different algorithms, each with their own strengths, weaknesses and abilities or capabilities for whichever target solution the designer is investigating, there is no one correct approach for modelling behaviour and social intelligence into artificial intelligence agents. As covered in Ad-Hoc Behaviour Authoring 2.1.1, *Finite State Machines (FSMs)* and *Behaviour Trees* are the most commonly used design approaches for designing NPCs in video games, but they lack the adaptability of unseen scenarios, unless the transitions are driven by indirect rule sets. Taking inspiration in *Charles Darwin's* theory of evolution and *Natural Selection*, a generational approach such as *Genetic Algorithms* or *Reinforcement Learning* presents themselves as likely candidates for developing and evolving complex behaviour.
Investigation into how *Intelligent Hives* such as flocks, colonises or packs communicate, can be used to help design intelligent systems. Inspiration can also be drawn from *Self-Adaptive & self-Organisation algorithms*. The flocking algorithm is an example of how simple rules such as *Avoidance*, *Separation*, and *Alignment*, to name a few, and communication between the agents can produce complex behaviour. The communication methodology of the *Ant Colonisation* algorithm is another indirect signalling between agents that can be considered for this project. Both of the two previously mentioned algorithms can be considered to be *Mimicry Designs*, copying or modeling based on properties of a target object or organism to improve the design in a way.

2.1.6 Limitations of Artificial Intelligence Agents in Video Games

The most common practice of modeling behaviour in NPCs in video-games is through using Ad-Hoc systems such as *Behaviour Trees*, *Finite State Machines*, and *Utility AI*. A major limitation of these approaches are the limited action spaces that the NPC will be able to acted within, is the in-ability to adapt appropriately to individual players' or other NPC behaviour, resulting in predictability and unrealistic behaviour.
Machine Learning, a wide subset of Artificial Intelligence, involves systems that are able to learn and improve from data without explicitly programming the desired behaviour. Machine Learning allows agents to examine a vast set of possible approaches to solve / optimize solutions for problems, which had not initially been considered by the designers, or those situations where overwhelming amounts of data severely restrict the programmers ability to directly solve complex and complicated variable parameters. Research into the field of improving behaviour modelling has shown that there is an untapped potential for using machine learning to develop new behaviours for autonomous agents

"This paper supports the hypothesis that incorporating ML in autonomous software agents allows for richer behavior in complex environments.[Roessingh et al. 2017]".

Exploration Exploitation Dilemma is a problem in data driven decision making processes, such as reinforcement learning. The problem is centred around balancing agents' exploitation of already known and explored patterns in their action-space in relation to their state, vs exploring new original possibilities. Exploiting already know solution-spaces generally means that the learning agents will repeat many previous actions leading to solutions that have already been found. but fail to explore new solutions, which can lead it to be stuck in local minimum/maximum.

Under-fitting & Over-fitting
One of the most apparent problems within machine learning is the problem of under and over fitting a model. To elucidate this, I will use a simple *Convolutional Neural Network (CNN)* for image classification of different flowers. Under-fitting a model is generally easier to detect, as the model would have a low accuracy in its classification. The most common reason is due to a too small data-set or an inadequate data-set that does not represent the target flowers sufficiently for the neural network to identify and learn the underlying patterns. Artificially creating a larger data-set is a potential option using *Data Augmentation*, which is the process of modifying elements in the data-set by various operations such as; *Translating, Rotation, Scaling, Shearing, Contrast balance, noise, etc.*. While *Data Augmentation* still needs satisfactory representations of the target classes (Floral Types), it helps train the model to correctly identify the classes under various conditions, making the model more stable.
Over-fitting of a model can happen if a model is trained on a too small data-set, or for too long. This results in the model not being able to adapt to new information that it has not seen before. Similarly to under-fitting, a larger more varied data-set and *Data Augmentation* are commonly approaches to solve the problem.

Limitations of Artificial Intelligence Agents in Video Games Conclusion
Artificial Intelligence agents in games are often too predictable and exhibit unnatural behaviour due to their limited action spaces. Providing them with a broader spectrum of actions spaces and sub-goals could potentially solve this. The observable space of agents should be varied, and can to some degree be randomised to help prevent *Under & Over-fitting* of the agent models.

This will allow agents to perform their goals in new environments, as well as making them less predictable. While a completely procedurally generated environment is not feasible for this project due to time constraints, some randomisation of the environment is possible.

2.2 Behavioural Psychology

With the goal of behavioural patterns emerging from the agents for this project, a brief investigation into *Behaviourism* will be used to allow a more definite classification of the emerging patterns. There are different branches of behavioural psychology, and an ongoing debate as to which stimuli are instrumental in behavioural development of humans. This project will focus primarily on how behavioural psychology can be affiliated with *Artificial Intelligence* Agents, and how in particular the *Operant Conditioning* relates to *Reinforcement Learning*.
Behaviour in both humans and other animals has been shown to develop based on a number of antecedent stimuli, a stimulus that causes an organism to reduce negative punishments and maximize rewards. *Reinforcement Learning* is based directly on this approach through its trial and error design, allowing agents to observe their environment, and provide adequate rewards to the agents for correct and incorrect action-state pairs.
In their book *Human Givens, A new approach to emotional health and clear thinking*, Joe Griffin and Ivan Tyrrell separate needs into *Physical Needs* and *Emotional Needs*, while both can be considered for evolving behaviours, this project will limit itself to emulating a selection of the *emotional* needs.

Griffin and *Tyrrell* lists the main human emotional givens as:[Griffin and Tyrrell 2004]

- Security - safe territory and an environment which allows us to develop fully.
- Attention (To give and receive it)
- Sense of Autonomy and control
- Being part of a wider community
- Friendship and intimacy
- Sense of status within social groupings
- Sense of competence and achievement
- Meaning and purpose - which come from being stretched in what we do (create) and think.

Figure 1: Emotional Givens

As the prototype takes the form of a simulated game, the given of *Security* is abstracted into increasing the survivability of the agents. By providing agents with an advantage if they are near something that increases their feeling of security then feeling more secure could potentially result in grouping behaviour. Incorporating a *Leader* agent into the game that buffs nearby allies will be considered for the development of the NPC agents.

2.3 State of the Art

This section will cover a selection of games that use some form of artificial intelligence design for their NPCs. As the documentation is not in the public domain, to avoid the possibility of exposing these companies' secrets, I will in this state of the art analysis try to generalize the AI for games where a direct source of their implementation techniques could not be identified.

2.3.1 Monster Hunter: World - Action Role Playing Game

I was unable to trace any direct documentation about the AI in *Monster Hunter: World*. An interview with the producer *Ryōzō Tsujimoto* and director *Yuya Tokuda* can however give some insight into their design.

> *"There's constant surprises - not just for players, but also for us. We've set up the rules for how they should behave, but in that sense we didn't script monster behaviour - we just gave them rules to behave by, and we still get surprised by how they respond to those rules."*
> [Donaldson 2017].

This answer suggest that enemy agent AI utilizes a type of behaviour tree with very abstract behaviour transition conditions. The different monster types in *Monster Hunter: World* each have their own set of different sensors, aggressiveness and patrol routes, which at times can overlap and cause infighting between the different enemies in the game. Some enemies, after having taken a certain amount of damage, divert from the fighting behaviour into a type of *Exhausted/Flee/Escape* state, where they will retreat and return back to their nest to recuperate. Populating the world of *Monster Hunter: World* with a variety of different monsters that each adhere to basic rule-sets is what allows different behavioural patterns to emerge in the game. However, the behaviour of the monsters is not completely unpredictable, players can learn the different characteristics of each enemy, how they interact with other monsters in the world, allowing players to develop as a monster hunter through learned experience.

2.3.2 Horizon Zero Dawn - Action Role Playing Game

In *Horizon Zero Dawn*, the enemy machine NPCs uses an AI system called a *Hierarchical Task Network Planner (HTNP)* [Thompson 2019]. This planner generates plans comprised of action macros, where each macron contains multiple actions in a set sequence. Through *HTNP*, the NPC agents can request for a plan of action to resolve a specific task, such as how to attack the player, running away, simple path-finding or investigating a disturbance in the environment, see Figure 2. Each machine NPC in *Horizon Zero Dawn* can function both as its own individual agent, or as a child of a group agent. This group agent keeps track of which agent children are a member of and it is able to generate plans for the group as a whole or the individual agents based on the situations requirements [Thompson 2019]. The group agents are also able to form new subgroups or destroy existing ones if the members of its group change, or new lone machines, which match the goals of the group agent approach. The members of the group agent are able to share information through what is known as a group *Blackboard*, where agents pass on information, such as, local patrol routes, valid locations, nearby disturbances, enemy locations etc [Thompson 2019]. When agent groups are in direct combat with the player, the attacks of the individual agents are based on a system that determines how interesting the attack would be for the player, dependent on a variety of factors such as whether the player can see the agent, how much damage they have dealt to the player already, and how much damage

has been dealt to them already by the player [Thompson 2019]. This is used to create variety in the combat, so the same agents do not repeatedly spam the same attack, and allow for other agents to take part.

Figure 2: Horizon Zero Dawn - Machine Actively alert and investigating nearby disturbance [Gilbert 2017]

2.3.3 Soulsborne Franchise

No direct documentation on the AI enemy agents for the *Soulsborne* franchise were found, and as a result, this section will cover a subjective analysis of how I think they were designed. The *Soulsborne* franchise AI system design covered here will look at the titles by **FromSoftware**: *Dark Souls 1*, *Dark Souls 2*, *Dark Souls 3*, *Bloodborne* and their most recent title *Elden Ring*. As there is an 11 year gap between *Dark Souls 1* and *Elden Ring*, the agent design has probably evolved, and thus a rough generalization will be made of the agent's behaviour, and how it is linked into gameplay.
The AI in the *Soulsborne* franchise are most likely designed based on *Ad-Hoc Behaviour* design models. The agents are aware of the types of attacks that the player uses, and react based on a variety of sensors which can be auditory and visual. A concrete example of this could be where an enemy has a shield, they are more likely to raise it to block if the player is attacking frequently. Other examples are seen in bosses which have more advanced reactions to play actions such as the Tree Sentinel in Figure 3b, which has a reflect magic spell attack that the boss only uses should the player actually use spells against them. It is also very likely that some NPCs directly have access to the player's inputs, the *Godskin* enemies in *Elden Ring* has been criticised heavily for this, as they cast a range attack the instant a player heals. Contrary to the design methodology in *Horizon Zero Dawn*, the enemies in the *Soulsborne* franchise do not wait for their turn to attack, and can easily and will gladly overwhelm the player. An essential design aspect of the *Soulsborne* franchise is however, that the player fails often, and can learn from mistakes. As one's character gets stronger by levelling up at bonfires, the majority of the enemies around the world respawn in the exact same location

every time. In this way the player learns gradually where the enemies are, and can develop a plan on how to defeat them more easily the next time they play, providing not only progress for the player's character, but also development of the player's understanding of the game itself.

(a) Dark Souls 1 Gameplay [Dan Curtis 2018]

(b) Elden Ring Gameplay [Corden 2022]

Figure 3: Dark Souls 1 & Elden Ring Gameplay

2.3.4 Left 4 Dead - Survival Horror Shooter

Left 4 Dead uses what is known as a *Director AI*, a system that attempts to direct the flow of the gameplay for the players based on the state of the game, managing the intensity of play so that it peaks and troughs [Greenwood-Ericksen 2010]. The director modifies the world around the players to keep the experience interesting and exciting by adding zombies to the map as the players progress. The *Director AI* determines when it should change the state of the game depending on the perceived *Stress level* of the players, this stress level is determined by a number of factors, such as the number of zombies attacking the players, attacks within proximity and special zombies attack on the players. As the game can be played with up to four players, each player has an individual stress level, and the director will directly target individual players if their stress level is lower than the others [Greenwood-Ericksen 2010].

Figure 4: Left 4 Dead Gameplay - Period with low stress level [mixtape_maker 2019]

2.3.5 Alien Isolation - Survival Horror

Alien Isolation is a survival horror game, where the main antagonist of the game is the alien known as a *Xenomorph*. The *Xenomorph* AI of *Alien Isolation* has been praised for its apparent intelligent behaviour, which is a result of two separate AI design systems, a *Macro AI* or *Director AI* and a *Micro AI*. The *Macro AI* manages the scene, the state of the game and the current status of both the player and the Alien. The *Micro AI* is the *Xenomorph's* direct AI, and determines the actions of the agent through a reactive, sense-driven behaviour tree, which reacts to both player actions and commands from the director. [AIandGames 2020]
The *Director AI* attempts to create varying moments of tension in the player based on what is referred to as a *Menace Gauge*. The gauge attempts to measure the intensity of the gameplay at a time, depending on the distance between the alien and the player, or whether or not the player can hear or see the agent. After a time of high menace levels, the *Director AI* will command the alien to leave the area or enter the vents to let the player have a break and will also to set traps for the player [AIandGames 2020]. After a break period as the menace gauge has lowered, the *Director AI* will prompt the *Xenomorph* to emerge again, and it will investigate, without giving the player's exact location to remain fair, see Figure 5. This type of behaviour is designed to create tension for the player, and this is directly reinforced through the design of the *Xenomorph* from a gameplay perspective as it can not be killed, and will one-shot the player if they are caught. [AIandGames 2020]
Some behaviours in the behaviour tree are initially locked off, unlocking as the player progresses throughout the game as a consequence of the players own behaviour [AIandGames 2020]. This allows the AI to exhibit new behaviours and creates variability in the gameplay, leading to less predictable behaviour. This gradual evolution of behaviour in the game ensures that the player's ability to play the game improves concurrently with the challenge of the *Xenomorph*.

This aligns with Mihaly Csikszentmihalyi's flow theory of keeping a balance between player skill and difficulty.

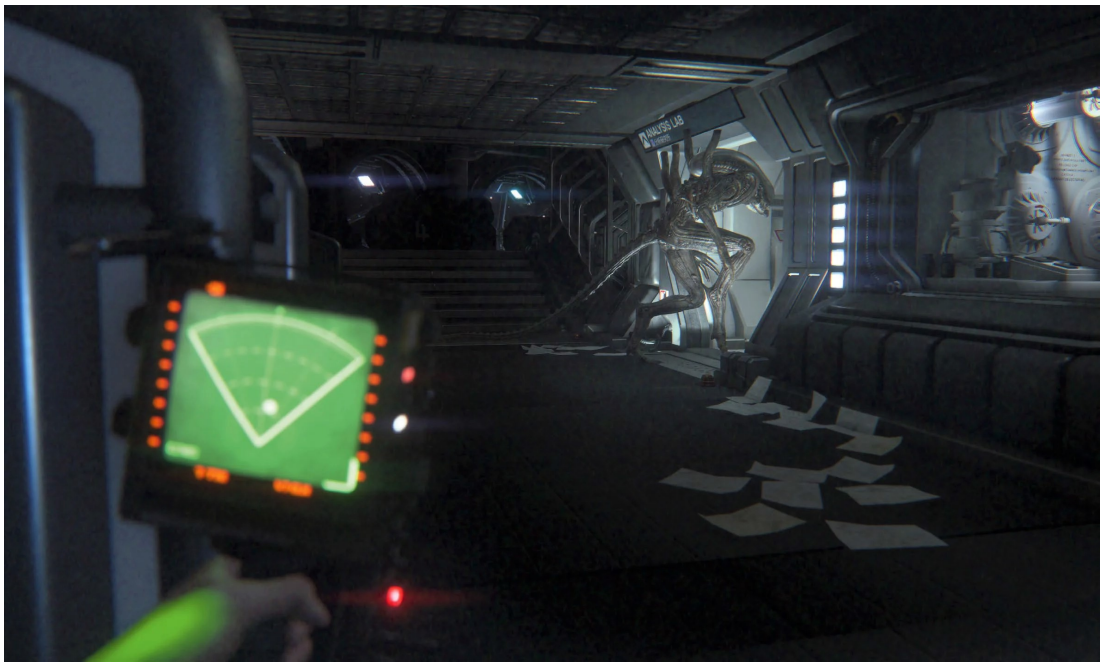


Figure 5: Alien Isolation - Alien searching rooms for the player [Rodriguez 2014]

2.3.6 State of the Art Conclusion

When I looked at a selection of different AI design systems, a number of design choices emerged which could be considered for the prototype for this project. The AI agent should not directly be designed with the objective of winning against the player, but instead have the objective of ensuring that the game feels fun, this can be done by varying the intensity of the game experience by adapting to the players behaviour, or by becoming more difficult over time as the player becomes more adept at playing the game. *Horizon Zero Dawn* does this by making combat and agent groups more exciting by variability in the enemy groups composition and attack patterns. Variability is an essential aspect of game design to make the game less repetitive. The different types of machine enemies also reinforce this by having different weaknesses and possible approaches on how to defeat them. We see in games such as *Horizon Zero Dawn*, *Alien Isolation* and *Left 4 Dead* that parameters used to change the behaviour of the AIs through a type of *Director AI*, *Interest Attack Gauge*, *Menace Gauge* and *Stress Level*. Adjusting the behaviour of the AI to ensure good game-flow is integral and vital in achieving the vision of the specific game. The agents for this project will however not consider this element due to time constraints.

2.4 Analysis Conclusion

With *Ad-Hoc Behaviour* AI designs, such as finite state machines and behaviour trees, as the most commonly used Agent design approaches, designing the baseline/control group for the test of this project will increase the ecological validity of the test. Behaviourism theory suggests that the reward system of correct and incorrect actions in *Reinforcement Learning* should be able to create behavioural changes in the agents depending on their environment. The goal of the agents should not be to win, but to ensure that the game entertains the player. We see examples of this design approach in state of the art games such as *Horizon Zero Dawn*, *Left 4 Dead* and *Alien Isolation* which consider the game experiences of the player when directing or commanding the enemies to ensure good game-flow. Players play games for a large variety of reasons; Community, story, challenge, discovery, etc. As such, there are a vast array of different factors the AI agents can consider when evaluating which behaviour it should choose from.

3 Methods

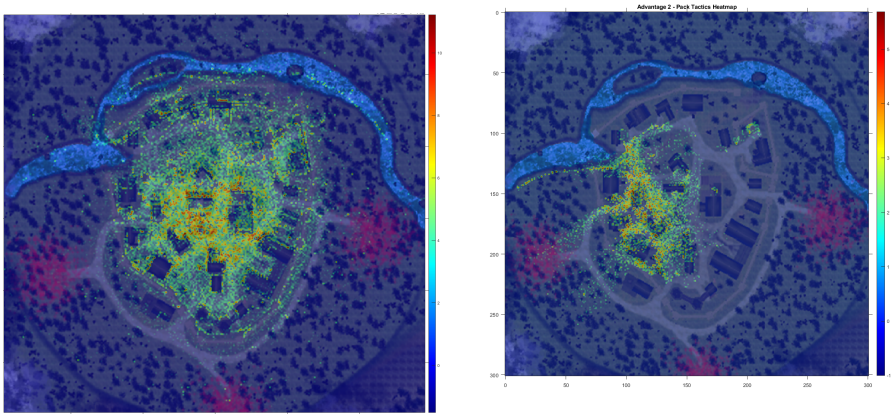
The following will present the methodology used for how data will be collected and analyzed. The aim of the project is to ascertain whether new types of behaviours can emerge through the use of indirect advantages in Artificial Intelligence agents. Two separate types of data will be gathered:

- The positional data of the agents to identify behavioural patterns over time.
- The numerical score of each agent to evaluate how they performed in the game setting they have been placed in.

3.1 Data Collection & Analysis Tools

3.1.1 Heatmap Test

Heatmaps are used to identify any patterns in the behaviour of the agents, the world position of each agent in the prototype is recorded and sent to a Json file created on initialisation. From this a heatmap of the agents were created and overlaid on top of the world-map. In Figure 6a the heatmap of the monster agent can be seen, here x ghouls competed against y NPC agents over a duration of 12 hours. This heatmap can be seen in Figure 6b. The use of heatmaps was partially inspired by *Gradient-Weighted Class Activation Mapping(Grad-CAM*, which is used primarily in classification networks to highlight important areas of an image which can help to evaluate whether the layers of the network are learning to identify relevant features.



(a) Monster Position Heatmap (b) Friendly NPC Heatmap

Figure 6: Heatmaps of the agents over a period of 12 hours

To evaluate changes in the behaviour of the agents through this heatmap methodology, two distinct maps will be made for each agent. The first will consider all of the agent's positions over a play duration and this will give an estimate of how often agents spend in the different sections of the time. The main goal of both agents is either the destruction or protection of the HeartTree in the middle of the village. I predict that the immediate area surrounding the HeartTree will be the area with the highest positional density for each group. Additionally, another heatmap will also be generated to show the average position of NPC groups larger than three. This is to ascertain whether the agents show a tendency to congregate when different advantages are included.

3.1.2 Score Test

The score of the individual agents will be provided depending on how well they are able to perform the following *Main & Sub - Goals*, Figure 7 presents an example of how the scores are added for the Friendly NPC agents. These scores are the same as those used as rewards for the agents during training and they are designed to maximize their score.

Objective Score List		
Goal Type	Objective	Reward / Punishment
Main	Protect Tree	10
Main	Kill Monsters	3
Sub	Avoid death by monsters	-10
Sub	Avoid walking into obstacles	-0.1

Figure 7: Example: Friendly NPC Agent Scores Test Rewards

3.2 Hypothesis

Based on an analysis of State of the Art games, see State of the Art 2.3, we believe that one approach is to make the agents less predictable and for them to act more diverse. To achieve the aim we will implement a method of communication between agents so that they can learn to cooperate and communicate their intentions. From this assumption, the following hypotheses were constructed:

Null Hypothesis: Agents when provided with indirect cooperative advantages will act predictably.
Alternative Hypothesis: Agents when provided with indirect cooperative advantages will behave less predictably and in a more diverse manner.

4 Design

This section consists of the design plan and process for developing the prototype and the testing strategy, see Methods 3. The design choices are derived from the findings from the analysis, see Analysis 2. The requirements for the prototype will first be outlined and this will help ensure the direction of the game and ensure that it can be adequately tested. Following a description of the fundamental *Game and & Level Design* elements will be presented, which will guide the development of the *Artificial Intelligence Agents*. The AI agents planned goals and functionality will then be presented, including their indirect advantages as well as the results of their overall performance. Specifics on the complete implementation of this prototype will be found in the subsequent section, see Implementation 5.

4.1 Design Requirements Overview

Based on Picard's assessment that a scenario, in which players are able to *role-play* and act out an emotional situation could potentially serve to expand agent's emotional dynamic range, an *RPG* genre for the game was chosen[Picard 1995]. Two different agents will be designed in an adversarial relationship, *Monsters* and *Friendly NPCs*, with the latter further divided into two separate models as the target of the hypothesis.
To test the hypothesis presented in Hypothesis 3.2, the design of the project should adhere to an array of different functional and non-functional design requirements. To begin designing the system, narrowing down the type of gameplay is necessary. Drawing inspiration from State of the Art RPG games that uses *Real-Time (RT)* combat systems like *The Witcher 3 (2015)*, *Final Fantasy XV (2016)*, *Monster Hunter: World (2018* and *Elden Ring (2022)*, presenting the player and AI agents with both offensive and defensive options is a necessity. Combat systems should be designed for both the player and the Artificial Intelligence Agents. While these games also allow the players to choose and tailor the combat style to a certain extent in order to fit their own personal preferences. The arsenal of the playable character is limited for this project due to time constraints, but could be expanded for further research into this area.

4.1.1 Functional Requirements

- Must be designed with elements from the 'RPG' game genre.
- Must contain a combat system for the player.
- Must contain a combat system for the Friendly NPC Agents.
- Must contain a combat system for the Hostile Agents.
- Must contain two separate agent types for the Friendly NPC Agents. Finite State machine & Reinforcement learning.
- Must contain a world environment that does not take the user out of the experience.

4.1.2 Non-Functional Requirements

- The prototype should have a maximum completion time of 20 min.
- Must give at least a low degree of engagement to the players.
- Must raise the difficulty of the game over time to ensure player flow.
- The control scheme should be easy to learn and understand.

4.2 Game & Level Design

Game design is an intricate mix of different elements that together form a complete picture. With a free design process, and a vast array of different genres, games can take many forms. This section will cover an analysis of certain aspects of game design taking into account the game world, game mechanics and the game as a whole for this project.

To adhere to state of the art *RPG* games, the game will be designed in a 3D space. Limiting the area that the agents will be able to explore should accelerate the training time of the project, while still offering the opportunity to create a story context for the actions of the player and the Friendly NPCs. An enclosed village will be designed to limit the play area. Buildings around the village will present the agents with obstacles that they have to learn how to circumnavigate. The main goal of the game for the hostile agents, the - *Monsters*, will be an attack on the village, with the aim of destroying an object in the centre. The main aim of the game for the *Friendly NPCs* will be to defend the target object. Further characteristics of the *Monster* agents, *Friendly NPC* baseline agents and *Friendly NPC* complex behaviour agent can be found in Monster Agent Characteristics - Reinforcement Learning 4.3.1, Friendly NPC Agents Characteristics - Baseline - Finite State Machine 4.3.2 and Friendly NPC Agents Characteristics - Reinforcement Learning: Advantages 4.3.3 respectively.
Player engagement is an essential aspect of game design, and encompasses a range of different elements. Clear feedback systems can help ensure that the player is able to understand the results of their environment and their own actions. To this end animations will be implemented for all agents, as well as particle effects for when an agent blocks, or when a character levels up.

4.2.1 Formal Elements

Games are formed by a constraint set by the rules of the system, Formal elements are the backbone of the game and form the overarching structure[Fullerton 2018]. Outlining these early on can help ensure a consistent vision for the game, in addition to shaping the rest of the design. Formal elements cover areas such as *Players*, *Objectives*, *Procedures*, *Rules*, *Resources*, *Conflict*, *Boundaries*, and *Outcomes*[Fullerton 2018]. Each element will be explored and defined throughout the design process, see Game & Level Design 4.2.

4.2.2 Designing Mechanics

Game mechanics are designed to follow the *rules* of the game, and allow *players* to reach their *objectives*. Inspiration from already existing games that fit within the target genre will be considered, and feature innovation to build on top of these mechanics while not diverging too far, should be the goal. An important element of designing mechanics is to ensure that the users receive adequate feedback of their actions, both for human players as well as for AI agents. Designing the game mechanics with this in mind will make it easier for both the players and agents to understand the impact of their actions. For human players, visual and auditory feedback are most commonly used, while for AI Agents, an observable parameter could be considered.

4.2.3 Designing Levels

When designing game levels, considering the *Size*, *Obstacles*, and *Objectives of level* are just a few of the elements that are paramount in designing a good level. Due to the limited scope of the project a smaller level size, with a limited number of obstacles is the chosen environment for the artificial intelligence to be trained.
Establishing certain rules for a procedural generated world could also have been a consideration, as that would ensure that the AI agents train in a non-static environment, preventing over-fitting of the agent models' observation space on environment data.

4.2.4 Progress and Variety

A core element of design within many games and role playing-games in particular is the concept of progress. Progress can be categorized into different types of goals for the player, long term goals, medium term goals, and short term goals. A common indication of progress in RPGs is the ability for the player character to get stronger over time. This can be in the form of levelling up the player character or improving their equipment. Many games also require that the player's skill level grow in conjunction with their avatar. A player normally gets better at playing a game by overcoming obstacles, as their understanding of the game's mechanics and systems increases. *Twitch-Mechanics* is a term encompassing gameplay where the players physical skills such as reaction time are in focus. Primarily Real-Time games such as those from the *Soulsborne* franchise fall under this category. On the opposite side are many *JRPGs (Japanese Role Playing Games*, that put a higher emphasis on the growth of the player's character.

4.2.5 Gameplay Engagement

Engagement in video games is a topic that a number of different studies have examined. Some of these research papers consider a combination of elements, such as flow, immersion, enjoyment, fun, presence, and intrinsic motivation[Abbasi et al. 2017][Schønau-Fog and Bjørner 2012].
While the topic of engagement in video games is an area that has been explored before, this project hopes to design artificial intelligence agents which can help maintain the player engagement whilst playing the game within the genre of Single Player Role Playing Video games through complex behaviours.

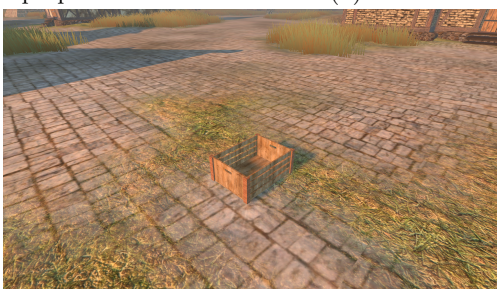
Curiosity and motivation to play, are examples of elements that have been studied with regard to engagement in video games. *'A Motivational Model of Video Game Engagement'* [Przybylski et al. 2010] discusses the effect of both intrinsic and extrinsic motivation for engagement, and assert *"Research has also shown that these forms of motivation have very different influences on individuals engaging in the same activity"*[Przybylski et al. 2010].
As the elements of motivation appear to be individually different, one can logically assume that the elements of engagement will also differ.
Mihaly Csikszentmihalyi's flow theory proposes that a balanced relationship between game difficulty and player skill level is essential for a fun experience.The game should strive towards ensuring that the challenge for the player is adequate and reasonable. Raising the difficulty of the game by gradually increasing the number of enemies could prove effective in ensuring that the player is kept engaged over a longer period of time.

4.2.6 Game World

The game world is designed with a classic medieval fantasy setting in mind. This ensures a direction for the style and creation of environmental props around the scene to function as obstacles for the agents. To reinforce player engagement, this detailed and pleasing game world is known to facilitate and improve the immersion of the player. To achieve this, a great emphasis has been placed on ensuring that the visual elements and effects of the project are of a certain standard. Simple props were modelled and placed around the environment to make it feel less empty. Figure 8 shows three such props that were placed around the world.

(a) Environmental prop - Barrel

(b) Environmental prop - Crate 0

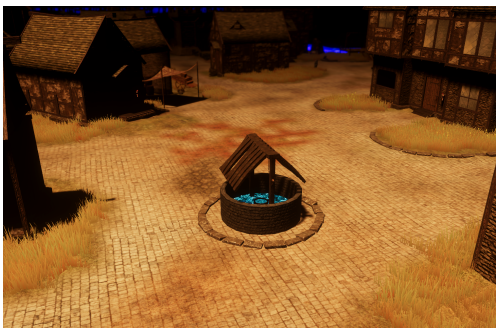


(c) Environmental prop - Crate 1

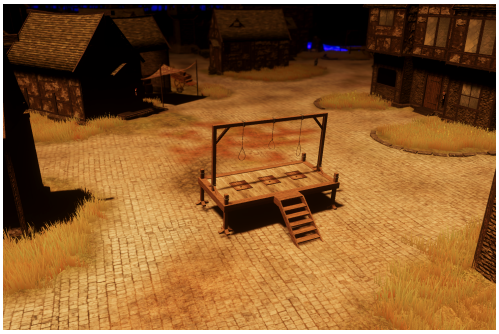
Figure 8: Environmental props

Bigger props were also designed to function as larger obstacles for the training agents and the cart in Figure 9 is an example of this. While scarcely placed around the world, the addition of these environmental props help make the world feel less empty, and therefore creates more variation in the game world.

Figure 9: Environmental prop - Cart



(a) Environmental prop - Well



(b) Environmental prop - Gallows

Figure 10: Environmental prop - Bigger Models

The original design of the village included a river flowing through the northern part of the village. This split that part of the village in two and necessitated that the agents had to cross a bridge. While the bridge seen in Figure 11a was modelled, the testing however showed that the environmental hazard within the village limits was too difficult for the agents, and severely increased the training time. As a result, the river was removed in the final design of the village. The Monsters were also initially set to spawn outside the village, in one of three possible graveyard areas that were walled off with an iron gate giving them an exit, see Figure 11b. This aspect of the game was also removed because of the complexity of first fulfilling two additional sub-goals namely: *1. Leave the graveyard*, *2. Breach the inner walls of the village*, before being able to train for their main-goal.

(a) Environmental prop - Bridge

(b) Environmental prop - Iron Gate

Figure 11: Environmental prop - Scrapped Models

The vast majority of the models of this project fall under the category of hard-surface models. The Tent Overhang model seen in Figure 12 was modelled to add some variety to the style of the game world. As it is also only used in one area of the world, it also functions as a landmark for the players, even though the village is quite small.

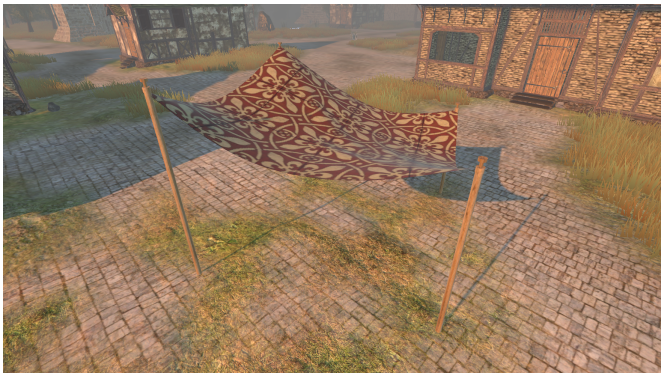
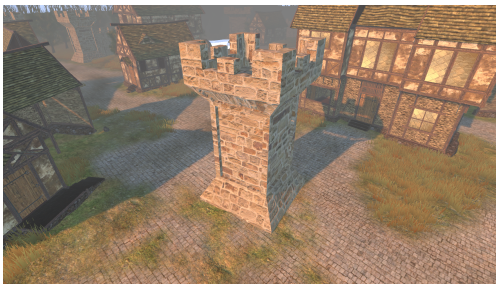


Figure 12: Environmental prop - Tent Overhang

Simple walls were designed to frame the village, these can be seen in Figure 13.

(a) Environmental prop - Straight Wall



(b) Environmental prop - Wall Corner

Figure 13: Environmental prop - Wall Pieces

The project features a day-night cycle, which is used not only as an aesthetic element, but also to control the game-pace. Monster agents are penalized for being inside the confines of the village limits during the day, but rewarded at night. The intention is to provide more variety in the game-flow, so the player can have a rest or go out of the village to hunt the monsters at their own discretion and pace. Figure 14 shows the lighting conditions for the different types of day.

(a) Day Lighting (b) Night Lighting

Figure 14: Environmental Lighting

Figure 15 shows a top down view of the game world. This map functions as a tool to help understand the movement patterns that emerge when training the agents. It functions primarily as a debugging tool, and to evaluate behavioural patterns.

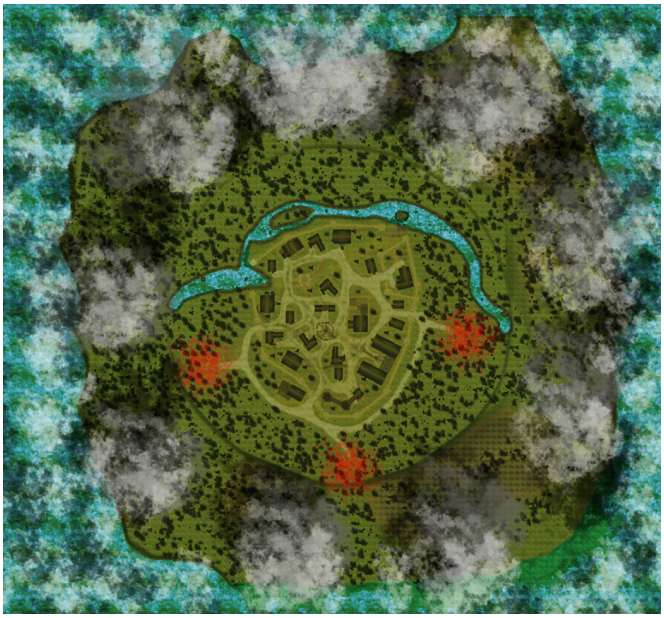


Figure 15: Environmental prop - World Map

4.3 Agents Characteristics

This section will cover the characteristics of the agents designed for this prototype. For the agents trained through reinforcement learning, the characteristics will include; the goals of the agents, their possible actions, observable variables, and what can ensure a possible reward or penalty to the agents' reward score. The characteristics of the FSM agent will present its goals, along with its possible states, transitions, conditions and actions

4.3.1 Monster Agent Characteristics - Reinforcement Learning

The monster agent will function as the adversary of the two types of Friendly NPC agents implemented. The model for the monster agent will be trained using reinforcement learning against the baseline finite state machine model to improve training times. As the behaviour of the monster agent is not the primary focus of this project, but rather to have a function of an obstacle to the Friendly NPC agents. Therefore, no communication methods between the monster agents will be implemented. This allows the agents to act independently without consideration of their fellow allies or for them to have the ability to change their strategy, regardless of how many allies they have. The goals of the agents are encouraged by reward scores, rewarding desired state-action pairs, and penalizing undesired ones.

Main Goal

- Destroy the HeartTree (Object in centre of the village) - Understand Prioritization of more dangerous enemies

The main goal of the monster agent is to locate and destroy the HeartTree at the centre of the village. This tree has a total of 500 health, and with an average base damage of the monsters of 30, it requires 17 hits for it to be destroyed. Each time a monster agent is on target and hits a tree it is rewarded with 0.2, by the game controller. When the tree is completely destroyed each monster agent receives a reward of 10.

Sub-Goals

- Kill enemy NPCs - Risk assessment of the danger of being near a enemies
- Avoid getting killed by enemy NPCs - Understanding of the danger of nearby enemies
- Avoid falling into the water - Understanding of environmental dangers
- Avoid walking into walls & obstacles - Understanding of Environment

The monsters have a variety of different sub goals that contribute to their total score. If they fall into the water around the edges of the northern part of the village or if they get killed by other NPCs, they get a penalty to their score, and they have to respawn and start a new episode. Walking into obstacles such as houses, trees, rocks or barricades reduces the agent's score by 0.1 for each frame, as well as dealing a small amount of damage. By using their raycast sensors, they should eventually learn how to avoid these obstacles. Killing the enemy NPCs adds a substantial reward of 1 to the agent's score. This is in addition to the rewards given for each successful hit. They do however take a risk, directly related to the proximity of the hostile agents.

Monster Agent Actions		
Action	**Possible Values**	**Action Type**
Move forward (Z-Axis)	[-1, 1]	Continuous
Turn around Y-axis	[-1, 1]	Continuous
Attack	True, False	Discrete

Total of 3 different actions

Figure 16: Monster Agent Actions

The monster agent has a total of three different possible actions, two continuous and one discrete. The continuous actions handle the forward movement of the monster as well as its rotation, meanwhile the discrete action only determines whether the monaster agent attacks or not. The action spaces of the monster agents, see Figure 16 were chosen as a result of the experience gained from multiple design iterations, and facilitated the ability to increase the training speed, while still ensuring that the monster agent could complete its main and sub-goals.

Monster Agent Observations		
Observation	**Size**	**Data Type**
Is the agent currently colliding with an object	1	Boolean
Is the agent currently attacking	1	Boolean
Is it currently day time	1	Boolean
Is the agent currently inside the village	1	Boolean
Current number of kills this section	1	Int
Dot Product of velocity and forward direction	1	Float
Dot Product of velocity and right direction	1	Float
Dot Product of forward direction and direction of incoming attack	1	Float
Inverse Euclidean Distance to the *HeartTree*	1	Float
Normalised Forward Direction	3	Vector3
Local space direction towards the *HeartTree*	3	Vector3
Normalised coordinates compared to the position of the *HeartTree*	3	Vector3
Normalised Euler Angles of the quaternion rotation of the agent	3	Vector3

21 total observations of the environmental state

Figure 17: Monster Agent Observations

For the monster agent to be able to achieve its goals, it needs to learn how to navigate the environment, and establish links between its current state, and the actions that yield the highest reward. To this end, the agent has a large observational space that ensures that it understands its location in the world. It also has to understand its orientation relative to that world, itself, the direction and distance from its main goal, the HeartTree. Additionally, the agent also continually receives information as to whether it is colliding with any obstacles and when it kills an enemy. The agent uses the observations to try to evaluate its current state, and consequently optimize its learning capacity and policy.

Monster Agent RayCast Sensors

Sensor Name	**Detectable Tags**	**Ray Info**
Monster NPC Detector	*Friendly NPC* *Player* *HeartTree*	5 rpd - FOV: 70 Degrees Length: 20 Units
Monster Water Detector	*Water*	5 rpd - FOV: 70 Degrees Length: 27 Units
Monster Obstacle Detector	*Obstacle* *Barricade*	3 rpd - FOV: 70 Degrees Length: 5 Units
Monster other Monster Detector	*Monster*	5 rpd - FOV: 70 Degrees Length: 20 Units
Monster Enemy in front Detector	*Friendly NPC* *Player* *HeartTree*	0∗ rpd - FOV: 70 Degrees Length: 2 Units

rpd: Rays per direction
∗ Default 1 ray forward
5 total RayCast sensors

Figure 18: Monster Agent RayCast Sensors

Visual sensors are attached to the monster agents that each provide information about its immediate environment. The observation size of the created observation can be calculated through $(ObservationStacks) * (1 + 2 * RaysPerDirection) * (NumDetectableTags + 2)$ [Elion 2019]. Due to the complexity of the environment, the total size of the observations gathered from the ray-cast sensors of the monster agent are 456. The world that the agent finds itself in is filled with a variety of different elements. Foes, houses, trees, rocks and barricades are scattered throughout the village, and the agent uses a total of five different sensors to keep track of what it can see within its field of vision, spanning 70 degrees directly in-front of it.

Monster Agent	
Reward	**Values**
Kills this Section∗	$\frac{Kills}{2}$
Successfully killed an enemy	0.5
Successfully hit Enemy	0.2
HeartTree killed	10
Inside the village during the night∗	0.001
Time Passing∗	Inverse Distance to HeartTree
Group Reward	
HeartTree is Destroyed	10
Penalty	
Unsuccessfully hit Enemy	-0.1
Colliding with an Obstacle∗	-0.1
Time Passing∗	$\frac{-1}{50000}$
Inside the village during the day∗	-0.001

∗ Added every frame
3 types of Individual rewards
1 type of group reward
3 types of penalties

Figure 19: Monster Agent Rewards and Penalties

Using the observations presented in Figure 17 and Figure 18, the monster agent attempts to maximize its possible reward score through the different actions presented in Figure 16. Figure 19 shows how the rewards for each individual agent are applied, rewarding the agent for correct actions, and penalizing it for incorrect ones.
An incentive to move towards the HeartTree was created for the monster agents by providing them with a reward based on a normalised value calculated by taking the reciprocal of the agents euclidean distance between the agents starting position and the HeartTree.

4.3.2 Friendly NPC Agents Characteristics - Baseline - Finite State Machine

A baseline of the Friendly NPC AI was created using the dominating artificial intelligence design methodology, *Finite State Machines (FSMs)*. This design provided the agents with three alternative states, *Patrol/Roaming*, *Chase*, and *Fight*. Transitions between the three states were conditioned on a Raycast spanning 120 degrees in front of their character and with 20 Unity units of magnitude. Their patrolling behaviour depends on navigating between various pre-set patrol points in the environment. To add some randomness to their routes, the agent choose a random patrol point to go to next after reaching its destination. Using Unity's Navmesh component provides the agents with orientational spatial awareness of the possible areas it could traverse, and thereby prevent colliding into any obstacles.

4.3.3 Friendly NPC Agents Characteristics - Reinforcement Learning: Advantages

Main Goal

- Protect Tree - Understand Prioritization of more dangerous enemies

Sub-Goals

- Kill Monsters - Risk assessment of the danger of being near a monster
- Avoid getting killed by Monsters - Understanding of the danger of nearby monsters
- Avoid falling into the water - Understanding of environmental dangers
- Avoid walking into walls & obstacles - Understanding of Environment

Friendly NPC Agent Actions

Action	Possible Values	Action Type
Move forward (Z-Axis)	[-1, 1]	Continuous
Turn around Y-axis	[-1, 1]	Continuous
Attack	True, False	Discrete
Leave Trail	True, False	Discrete

Total of 4 different actions

Figure 20: Friendly NPC Agent Actions

The Friendly NPC agent has a total of four different possible actions; two continuous and 2 discrete. The continuous actions handle the forward movement of the monster as well as its rotation, meanwhile the discrete actions determine whether the Friendly NPC agent attacks, or leaves a trail on the ground. The action space of the Friendly NPC agents, see Figure 20 was reduced from its original 8, which gave it the ability to block, run, and level up attributes. This decision was made to increase training speed while still ensuring that the Friendly NPC agent could complete its main and sub-goals.

Friendly NPC Agent Observations

Observation	Size	Data Type
Is the agent currently colliding with an object	1	Boolean
Is the agent currently attacking	1	Boolean
Does a nearby ally have Advantage 1 (Perception Range Bonus)	1	Boolean
Does a nearby ally have Advantage 2 (Pack Tactics)	1	Boolean
Does a nearby ally have Advantage 3 (AoE Healing)∗	1	Boolean
Number of nearby allies	1	Int
HeartTree's current health compared to its maximum health normalized	1	Float
Dot Product of velocity and forward direction	1	Float
Dot Product of velocity and right direction	1	Float
Dot Product of forward direction and direction of incoming attack	1	Float
Inverse Euclidean Distance to the *HeartTree*	1	Float
Normalised Forward Direction	3	Vector3
Local space direction towards the *HeartTree*	3	Vector3
Normalised coordinates compared to the position of the *HeartTree*	3	Vector3
Normalised Euler Angles of the quaternion rotation of the agent	3	Vector3

∗ Not used

21 total observations of the environmental state

Figure 21: Friendly NPC Agent Observations

In a similar manner to the Monster Agents, the Friendly NPC agents use a combination of direct observations, see Figure 21 and sensors, see Figure 22 to establish links between its current state, and the actions that yield the highest reward. In contrast to the monster agents, the Friendly NPC agents directly observe the health of the HeartTree, and due to the Friendly NPCs advantages they also need information on how many allies are nearby, and whether any of them possess an advantage they can benefit from.

Friendly NPC Agent RayCast Sensors

Sensor Name	Detectable Tags	Ray Info
NPC Character Detector	*Friendly NPC* *Player* *HeartTree*	3 rpd - FOV: 70 Degrees Length: 20 Units
NPC Water Detector	*Water*	2 rpd - FOV: 70 Degrees Length: 27 Units
NPC Obstacle Detector	*Obstacle* *Barricade*	3 rpd - FOV: 70 Degrees Length: 5 Units
NPC Enemy Monster Detector	*Monster*	3 rpd - FOV: 70 Degrees Length: 20 Units
Monster Enemy in front Detector	*Monster*	0∗ rpd - FOV: 70 Degrees Length: 1 Units
Trail Detector	*Trail*	2 rpd - FOV: 70 Degrees Length: 15 Units

rpd: Rays per direction
∗ Default 1 ray forward
6 total RayCast sensors

Figure 22: Friendly NPC Agent RayCast Sensors

Visual sensors are attached to the friendly agents and each provides different information about its immediate environment. The observation size of the created observation can be calculated through $(ObservationStacks) * (1 + 2 * RaysPerDirection) * (NumDetectableTags + 2)$ [Elion 2019]. Due to the complexity of the environment, the total size of the observations gathered from the ray-cast sensors of the Friendly NPC agent are 117. The Friendly NPC agent has an additional vector that allows it to observe the trails left on the ground by other ally agents.

Friendly NPC Agent	
Reward	**Values**
Kills this Section∗	$\frac{Kills}{2}$
Successfully killed an enemy	0.5
Successfully hit enemy	0.2 x damage / 10
Time Passing∗	Inverse Distance to Heart-Tree
Group Penalty	
HeartTree is Destroyed	-10
Penalty	
Hit by an enemy	-0.2
Unsuccessfully hit enemy	-0.1
Successfully hit ally	-0.2 x damage / 10
HeartTree killed	-10
Dying	-10
Colliding with an Obstacle∗	-0.1
Time Passing∗	$\frac{-1}{50000}$

∗ Added every frame
4 types of Individual rewards
1 type of group penalty
7 types of penalties

Figure 23: Friendly NPC Agent Rewards and Penalties

The agent receives a reward or penalty to its total score depending on a variety of actions presented in Figure 23. Rewards and penalties are tools to motivate the agent to complete their main and sub-goals. The monster agent receives a group reward when the HeartTree is destroyed, the Friendly NPC Agent on the other hand does not receive a reward if it survives, but avoids the penalty. The Friendly NPC agents are also penalized if they are killed, or if they hit their allies, punishing random attacks in the open more than the monster agents.

Designing Indirect Advantages Findings from the exploration of different games covered in the State of the Art chapter, see State of the Art 2.3, show that a frequently used communication method between agent groups is the *Blackboard*, a space where the agents send and receive information relevant to their current goal, for example the player location. Drawing inspiration from this idea, an indirect advantage could be designed based on a similar data communication system.
Four different indirect advantages have been designed for this project, and three will be trained. The resulting behaviour will be analyzed and tested for this project:

Advantage	Condition	Predicted Behaviour
Perception Range Bonus	Allies Nearby	Grouping
Pack Tactics	Allies Nearby	Grouping
Area of Effect Healing, Reduced Damage Output	Allies Nearby	Grouping, Agent Protection
Pheromone Communication	None	Area Camping

Figure 24: NPC Possible Advantages

Perception Range Bonus: The *Perception Range Bonus* advantage increases the ray-cast lengths of the sight sensor for the agents if there are allies nearby. This allows the agents to more easily identify threats and obstacles, enhancing their observational space. As only one other agent is necessary for the agents to gain this advantage, one could predict that groups of two or more might form.

Pack Tactics: Inspired by a common trait from *Dungeon & Dragons* of the same name, this advantage provides a 100% damage bonus for agents when there is an ally nearby. Similarly, to the *Perception Range Bonus*, only one other agent needs to be present nearby to activate this advantage. This could lead to groups of two or more forming and form a compound advantage.

Area of Effect Healing, Reduced Damage Output: This advantage provides a small healing area of effect affecting nearby allies, but taking advantage of this, reduces the damage output of attack by 50%. While this increases the survivability of nearby agents, the trade-off means large decrease in the possible damage per second, and might therefore still result in an overall lower score. Having a balanced ratio between the number of agents with this trait would allow the agent group as a whole to combat larger groups of enemies over a longer period of time. Additionally, if the agents are trained in a manner where these traits ensure survival over an extended period of time, the agents could learn to protect themselves and indirectly other agents by implementing and using this trait.

Pheromone Communication: Inspired by the Ant Colonisation algorithm: this possible advantage allows agents to leave a trail on the ground. This trail of earlier learnt experiences from environmental stimuli can enhance communication and possibly lead to evolving an intra communication network. No programmed behaviour or information about what the trail left means is provided thus giving the agents an opportunity to utilize this communication method however they see fit. This, therefore facilitates an unexplored area of development for this retrospective form of communication The agents can independently develop and possibly utilize this communication method to discover whether it gives an evolutionary advantage.

4.4 Design Conclusion

Both the *Monster* and the *Friendly NPC* agents have gone through numerous iterations, both in regards to their possible action spaces, which elements of the environment should be included in the observation space, or simply due to mistakes in terms of correctly designing the environment for the agents.
The advantages that will be tested for this project will be *Perception Range Bonus*, *Pack Tactics*, and *Pheromone Communication*. *Perception Range Bonus* and *Pack Tactics* will be

considered as the two main advantages and each of them will be trained separately, while *Pheromone Communication* will be applied to both models.
A simple yet pleasing visual setting has been created for the prototype for this project. This was possible with the help of a number of different Unity Assets, see Credits & Unity Assets 5.3. All smaller environmental props as presented in Figure 8 I created using *Autodesk Maya*.

5 Implementation

This section will go into detail regarding how some of the more critical elements of the prototype function and of how the agents and their environment were implemented. More than 60 scripts were written for this project. These range from changing the day-night cycle, to the full controller of the AI agents. This section will primarily focus on the Game World system and the Artificial Intelligence implementation. For all the scripts used in the project, see the attached game project folder attached to this thesis. As the scripts for the Monster Agents and Friendly NPC agents are designed in a similar fashion, there are many similarities between the scripts, and only code snippets specific to the agent in question will be covered.

5.1 Monster - Reinforcement Learning - Implementation

The agent model for the monster agent was trained against the Finite State Machine version of the Friendly NPC Agent. The background for this decision was to increase the training speed and to increase the efficiency of the training. The reason for this was because training two models simultaneously, resulted in the models training efficiency to be directly dependent on the success of the other. The final version of the model for the Monster agents were then used to help train the Friendly NPC Agents with the chosen advantages, *Perception Range Bonus* or *Pack Tactics*.
When a monster agent is set to spawn, the Game Controller randomly selects one of four possible monster Prefabs. They all share the same fundamental components, but have individual meshes and animation controllers. The four possible meshes can be seen in Figure 25. Credits for the models and animations can be found in Credits & Unity Assets 5.3.

Figure 25: Monster Meshes - From left to right: Giant Rat, Ghoul, Evil Watcher, Oak Tree Ent

The monster agent has many different attributes that define it. These are all applied on initialisation, see Figure 26, through a reference to a *Scriptable* Unity object. This allows for rapid changes in all instances of the agent, and all variants of the agent.

```
1       public override void Initialize()
2       {
3           SetResetParameters();
4           EnvironmentParameters = Academy.Instance.EnvironmentParameters;
5
6           //------ Set Starting Stats ------//
7           damage = cs.ghoulDefaultDamage;
8           lives = cs.ghoulLives;
9
10          maxHealth = cs.ghoulMaxHealth;
11
12          health = cs.maxHealth;
13
14          gravity = -cs.gravity;
15          groundDistance = cs.groundDistance;
16          turnSmoothTime = cs.turnSmoothTime;
17
18          walkSpeed = cs.ghoulWalkSpeed;
19
20          speed = walkSpeed;
21          latestDirection = this.transform.forward;
22
23          startingPosition = this.transform.position;
24
25          npcKillReward = cs.ghoulFriendlyNPCKillReward;
26          playerKillReward = cs.playerKillReward;
27          heartTreeReward = cs.ghoulHeartTreeKillReward;
28          barricadeReward = cs.ghoulBarricadeKillReward;
29          attackReward = cs.attackReward;
30          damageTakenReward = cs.damageTakenReward;
31          deathReward = cs.deathReward;
32      }
```

Figure 26: Code snippet - Monster Agent Script - Initialisation

The three functions shown in Figure 27 are the core of the monster agent. Here the action spaces are set for the agents two continuous actions and the discrete action. The *Move* and *Look* function controls the forward movement and rotation of the agent respectively.

```
1       public void ControlGhoul(ActionBuffers actionBuffers)
2       {
3           var discreteActions = actionBuffers.DiscreteActions;
4           var continuousActions = actionBuffers.ContinuousActions;
5
6           m_InputV = continuousActions[0];
7           m_Rotate = continuousActions[1];
8           Look(m_Rotate);
9           var moveDir = transform.TransformDirection(new Vector3(0, 0, Mathf.Abs(m_InputV)));
10          Move(moveDir);
11          int attack = (int)discreteActions[0];
12          if (attack == 1) { AttackTarget(); }
13      }
14
15      public void Move(Vector3 dir)
16      {
17          var vel = rb.velocity.magnitude;
18          float adjustedSpeed = Mathf.Clamp(runSpeed - vel, 0, runSpeed + 1);
19
20          rb.AddForce(dir * adjustedSpeed, ForceMode.Impulse);
21      }
22
23      public void Look(float xRot = 0)
24      {
25          m_Yaw += xRot * MouseSensitivity;
26          float smoothYawOld = m_SmoothYaw;
27          m_SmoothYaw = Mathf.SmoothDampAngle(m_SmoothYaw, m_Yaw, ref m_YawSmoothV, MouseSmoothTime);
28          rb.MoveRotation(rb.rotation * Quaternion.AngleAxis(Mathf.DeltaAngle(smoothYawOld, m_SmoothYaw), transform.up));
29      }
```

Figure 27: Code snippet - Monster Agent Script - Monster Controls

Through various testing sections, a variety of different unexpected behaviour of the monsters emerged. When the monster agents were outfitted with the ability to move both forward and backwards, they showed a strong preference for moving backwards. I speculate that it could be due to the fact that their attacks are frontal based, and with the protective capacity of a wall at their back, they are less likely to be ambushed by attackers.

5.2 Friendly NPC Agent - Finite State Machine - Implementation

The finite state machine baseline agent was designed with three different states; *Patrol*, *Chase* and *Fight*. The transitions between these three states are dependent on a sight sensor, which attempts to imitate the RayCast sensors which its reinforcement learning counterpart uses. If there is an enemy within the agent's field of view, the agent will change its NavigationMesh to find the quickest route to them, see Figure 28.

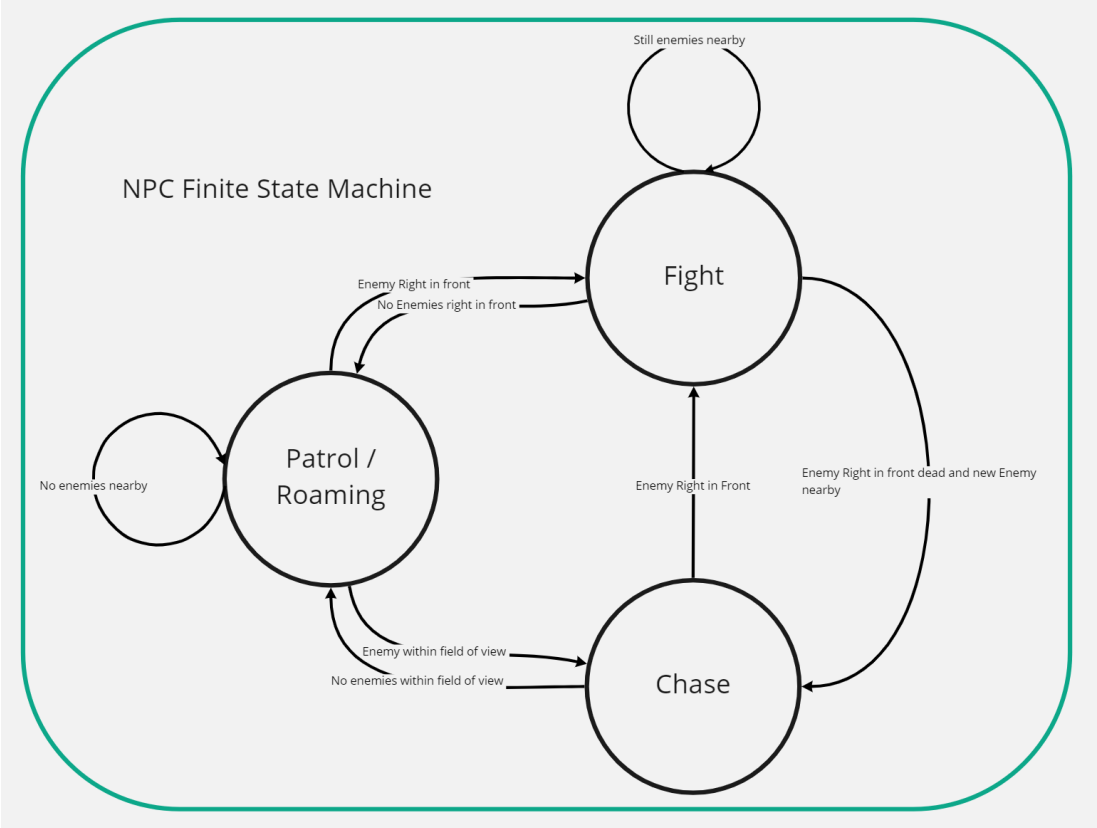


Figure 28: Finite State Machine of the Baseline Friendly NPC Agent

Should an enemy be directly in front of the agent, the agent will transition to the *Fight* State, where the agent will execute an action every 2^{nd} second such as *Attack*, *Block*, *Dodge* or nothing, with varying probabilities as presented in Figure 29. The *Patrol* and *Chase* stats of the agents simply direct which location the agents should move to next.

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


namespace P10.FSMNPCController
{
    [CreateAssetMenu(menuName = "FSM/Actions/FightAction")]
    public class FightAction : FSMAction
    {
        // Start is called before the first frame update
        public override void Execute(BaseStateMachine stateMachine)
        {
            var nearbyEnemies = stateMachine.GetComponentInChildren<PackTacticsChecker>().nearbyEnemies.Count;
            var nearbyEnemiesList = stateMachine.GetComponentInChildren<PackTacticsChecker>().nearbyEnemies;
            var mySightSensor = stateMachine.GetComponent<EnemySightSensor>();
            var myNavMesh = stateMachine.GetComponent<NavMeshAgent>();
            if (mySightSensor.targetTransform != null)
                myNavMesh.SetDestination(mySightSensor.targetTransform.position);

            if (nearbyEnemies != 0 && mySightSensor.EnemyCheckDistance())
            {
            var combatController = stateMachine.GetComponent<FSMNPCCombatController>();
            if(mySightSensor.EnemyAroundMe())
                {
                    myNavMesh.isStopped = false;
                    Debug.Log("There is an enemy Around me: " + stateMachine.transform.root.name);
                    myNavMesh.SetDestination(nearbyEnemiesList[0].transform.position);

                }

            float randomChoice = Random.Range(0, 50);
            if(randomChoice <= 10)
            {
                combatController.controller.AttackTarget();
                combatController.controller.blocking = false;
            } else if (randomChoice >= 11 && randomChoice <= 20 && combatController.controller.stamina > 20)
            {
                combatController.controller.Dodge(new Vector3(Random.Range(0, 359), 0, Random.Range(0, 359)));
                combatController.controller.blocking = false;
            } else if (randomChoice > 20 && randomChoice <= 25 && combatController.controller.stamina > 20)
            {
                combatController.controller.blocking = true;
            }
            }
        }
    }
}
```

Figure 29: Code snippet - FSM Fight Action Script

5.2.1 Player Controller & Friendly NPC Agent(Advantages) - Reinforcement Learning - Implementation

This section will cover both the Friendly NPC Agent and the Player Character as the original design provided both the same types of actions. The Friendly NPC agent implementation shares many similarities with the monster agent covered in Monster - Reinforcement Learning - Implementation 5.1. One of the main differences is the inclusion of area wide advantages to nearby allies, and an expanded set of actions. To increase the training speed of Friendly NPC agents, three actions were: *Barricades*, *Blocking*, and *Dodging*. These three actions will be explained as they are still available for the player's character.
I created a mesh for the NPC agents that allowed animations to be rapidly applied through animations from `Mixamo.com`. This allowed for quick prototyping of the game mechanics and feedback system for the actions the player and Friendly NPCs could take. Figure 30 shows the mesh that the Friendly NPCs and the player uses.

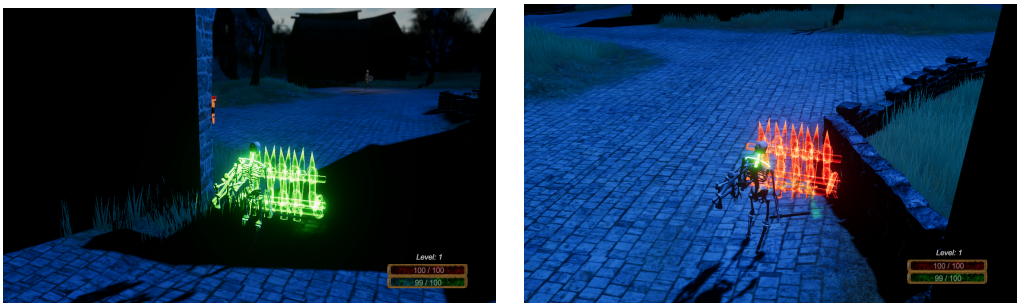
Figure 30: Friendly NPC and Player mesh. By Author

To test the agents, a heuristic control system for the NPC agents was implemented, and multiple test runs were conducted to identify bugs between the many systems in the prototype. This was programmed in tandem with the player controller to ensure that the Friendly NPC agent was presented with the same options as the player.

```
1                   if (Input.GetKeyDown(KeyCode.B) && canPlaceBarricade && !crouching && !animator.GetBool("Jumping") && ...
                        !animator.GetBool("Falling"))
2                   {
3                       placingBarricade = !placingBarricade;
4                       if (placingBarricadeInProgress)
5                       {
6                           DisableBarricadeMode();
7                       }
8                   }
9
10                  if (placingBarricade)
11                  {
12                      if (animator.GetBool("Jumping") || animator.GetBool("Falling") || animator.GetBool("Dodging"))
13                      {
14                          DisableBarricadeMode();
15
16                      }
17
18                      if (!placingBarricadeInProgress)
19                      {
20                          var tempBarricadeProjection = Instantiate(barricadeProjection, transform.position + ...
                                transform.forward * 2, transform.rotation, transform);
21                          tempGameObject = tempBarricadeProjection;
22                          placingBarricadeInProgress = true;
23                      }
24
25                      if (Input.GetKey(KeyCode.Mouse0) && placingBarricadeInProgress && ...
                            tempGameObject.GetComponent<BarricadePlacementChecker>().canPlaceBarricade)
26                      {
27                          if (tempGameObject != null)
28                          {
29                              Instantiate(barricadeObject, tempGameObject.transform.position, ...
                                    tempGameObject.transform.rotation);
30
31                              DisableBarricadeMode();
32                              availableBarricades--;
33                          }
34                      }
35                  }
```

Figure 31: Code snippet - Barricade Placement - Player Controller Script

The barricade placement function checks whether the character is currently in the *Placing Barricade Mode* and has an available barricade left. If these conditions are true, then the function instantiates a temporary barricade that is a child of the character model. The colour of the barricade then changes, depending on whether the temporary barricade collides with any other object. Green if the character is able to place the barricade and red if they are unable, see Figure 32. The barricade uses a custom written liquid shader that allows for transparency, illumination through a reflective bumpmap, and fresnel. This shader is also used for all water sources around the environment, see Appendix: B - Multi Purpose Liquid Shader 11.2.

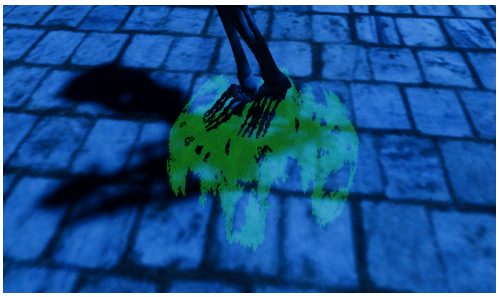


(a) Able to place barricade (b) Unable to place barricade

Figure 32: Placing Barricade Action

The Friendly NPCs had four fully functional advantages implemented that could be enabled

independently. The *Trail* advantage was enabled for both test groups, and allowed agents the option to leave behind a semi transparent trail on the ground that other NPCs can observe using their Raycast sensors, see Figure 33a. The trail material has a green emission so it is also visible for the player during the night.



(a) Trail left by Friendly NPCs

(b) Player Status Screen

Figure 33: Trail - Player Status Screen

The remaining three advantages *Perception Bonus*, *Pack Tactics*, and *AoE Healing* each made up of a function called *DoesNearbyAlliesHaveAdvantages*, which returns a Boolean array of size three. One Boolean for each of the possible advantages, see Appendix: C - Advantage Checker 11.3. The function initialises three Booleans when called, and has access to another script on the character that holds a list of allies nearby. The Friendly NPC agent and player controllers each have three Lists of GameObjects, one for each Advantage. The function then goes through each ally and if an ally has an advantage enabled and gets added to the corresponding GameObject List. The agent then checks the length of these three Lists of GameObjects to determine whether it gets the advantages buffs.

A level up system for both the player and the NPCs(Removed as an action in the final version) was implemented allowing the characters to increase their base stats and unlock advantages when they have reached a the necessary amount of experience, see Figure 33b. This system was originally designed to allow agents to shape their own development, which stats to prioritize, and which advantages provided the best results in given situations and team setups.

5.3 Credits & Unity Assets

The Monster models and animations are from the *HEROIC FANTASY CREATURES FULL PACK Volume 1* pack, see `https://assetstore.unity.com/packages/3d/characters/creatures/heroic-fantasy-creatures-full-pack-volume-1-5730`
All Friendly NPC animations are downloaded from *Mixamo*, see `https://www.mixamo.com/#/`
The houses in the village are from the *FREE Medieval Structure Kit*, found at `https://assetstore.unity.com/packages/3d/environments/fantasy/free-medieval-structure-kit-141700`
The skybox is from *Starfield Skybox*, see `https://assetstore.unity.com/packages/2d/textures-materials/sky/starfield-skybox-92717`
The Environmental props such as trees and rocks are from *Nature Starter Kit 2*, see `https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977` and *Environmental Asset Pack*, see `https://assetstore.unity.com/packages/3d/environments/`

environmental-asset-pack-170036

Many thanks to *Mikkel Bækmark Zoffmann* for providing a script I use for collecting the positional data of the agents. This data is used to create the heatmap of the routes of the NPCs used in Results 6
The background music playing throughout the game are from *Ultimate Game Music Collection*, see https://assetstore.unity.com/packages/audio/music/orchestral/ultimate-game-music-collection-373
Game World 4.2.6 showcases some of the different assets I made for the project

5.4 Implementation - Conclusion

The agents and player character for this project has gone through a number of design iterations, and because of time constraints many features had to be removed to speed up the training time. The game world is also filled with a large number of scripts to improve the visual quality of the game. The torches around the village flicker at random intervals at night, the skybox gradually transitions along with an increase in fog density and the primary Light object that rotates to create the sense of the sun setting and moon rising.

6 Results

This section will present the raw data of the results from the *Heatmap Test* and the *Score Test* as well as the data with outliers removed for the Monster Agents and the three different types of Friendly NPC Agents; *Finite State Machine Model*, *Advantage 1 (Perception Range Bonus)* and *Advantage 2 (Pack Tactics)*. Analysis of Results 7 will delve into a comparison of the different models based on the test data, and Test 8.2 covers a discussion on the shortcomings of the test, and how it could have been improved and designed differently. The raw data files used can be found in the appendix folder attached with this thesis.

6.1 Heatmap Test

6.1.1 Monster Agent Heatmap Test Results

After a training period of 30.000.000 steps, the monster agent had developed a behaviour that, while not completely optimal, was deemed adequate for functioning as an adversary to the training of the Friendly NPC Agents. Data of the positions of the agents over time were written to a Json file over multiple sections using the final model of the agent. A 3D histogram of the positional data from the agents was generated, see Figure 34, and outliers were identified and removed. All the tall bins seen were marked as outliers, as they were generally corners or other places in the environment where the agents could get stuck. See Appendix: A - MatLab Data Handling Script 11.1 for the MatLab code. The X and Z values of the agent's position that the graph uses were rounded to integers, as only a general idea of their position was relevant.

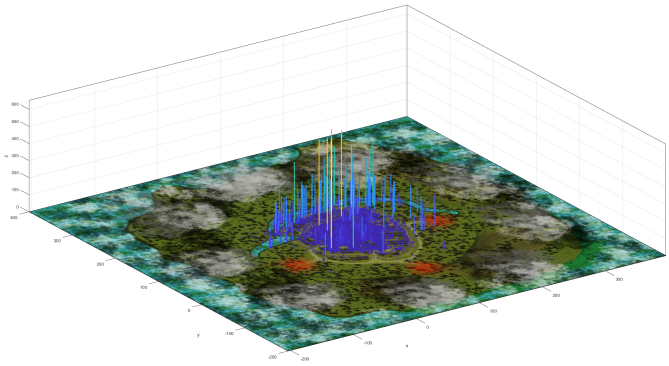
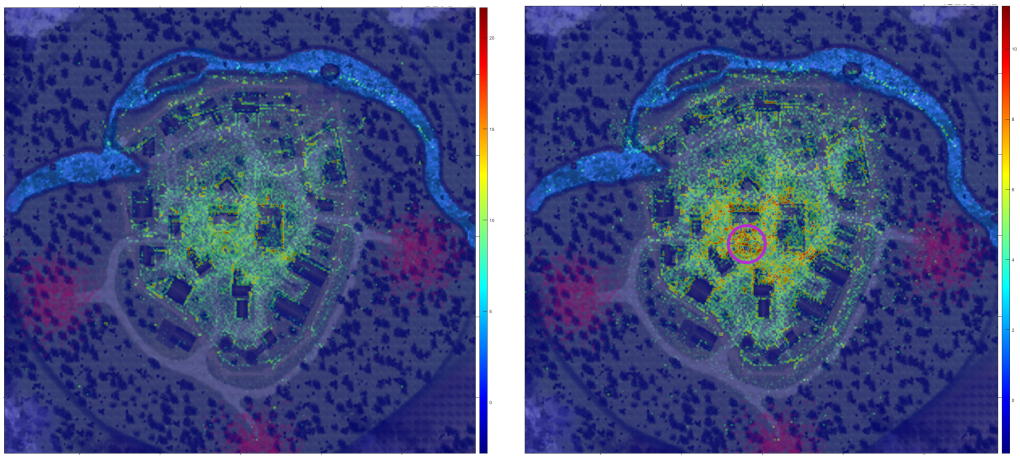


Figure 34: 3D Histogram of Monster Agents' positional Data

Using the data from the 3D histogram ,with outliers removed, a heatmap was generated based on the bin heights for the positions of the agents. This heatmap was again overlaid on the world map of the game environment in order to help identify areas of interest for the agents, see Figure 35.



(a) Raw positional data

(b) Outliers removed, HeartTree marked with a purple circle

Figure 35: Heatmaps of Monster Agents' Positional Data

6.1.2 FSM NPC Agent Heatmap Test Results

Data was collected for the positional data of 10 Friendly NPC agents that used the Finite State Machine model. Figure 36 shows a 3D histogram of the agents positions, with additional bins in the corners of the map.

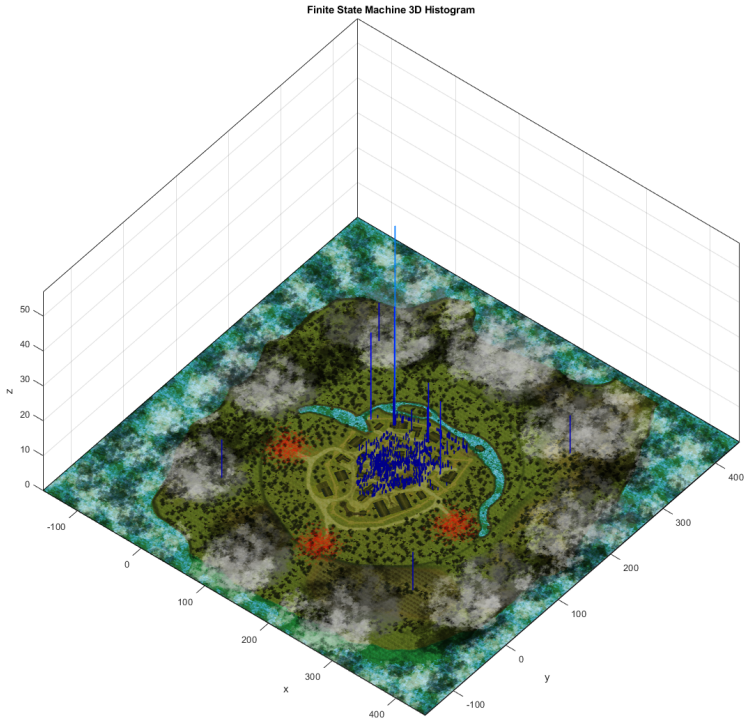
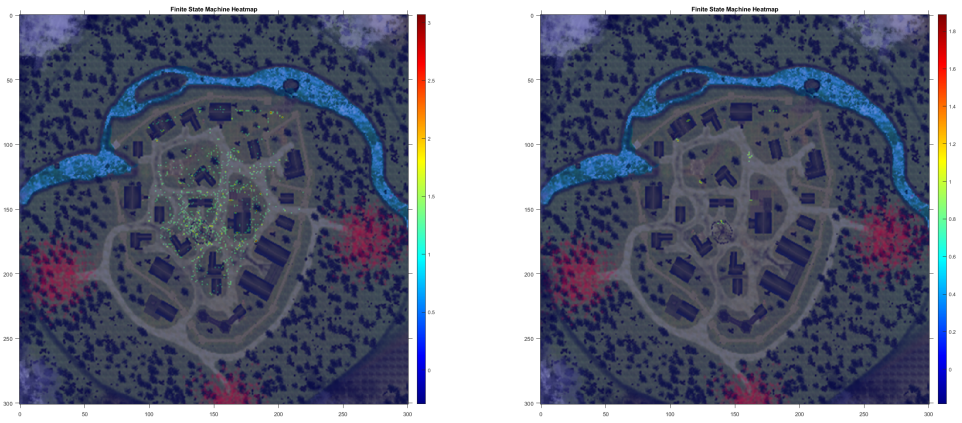


Figure 36: 3D Histogram of FSM Agents' positional Data

High bins were examined and identified as outliers. These were generally cases where an NPC had spawned inside a building and was unable to get out. The primary activity of the agents are between patrol points, which were placed where pathways connected. No area of the village shows a large amount of activity, with a smaller amount present near the HeartTree in the centre of the city. Figure 37b displays a heatmap of the agents activity when agents had more than two nearby allies. No distinct cluster can be identified.



(a) Heatmap of positional data with outliers removed for FSM Agents

(b) Heatmap of positional data with outliers removed for FSM Agents with 2+ nearby allies

Figure 37: Positional Data of FSM Agents

6.1.3 Advantage 1 - Perception Bonus - NPC Agent Heatmap Test Results

Data was collected for the positional data for the 10 Friendly NPC agents over a period of 14 hours. From these 10, 3 of them were outfitted with the *Perception Bonus* advantage. This advantage provides a buff to all nearby allies, increasing the length of the Raycast sensor components by 0.4 units for each nearby ally. Figure 38 shows a 3D histogram of the agents' positions, with additional bins representing the corners of the map.

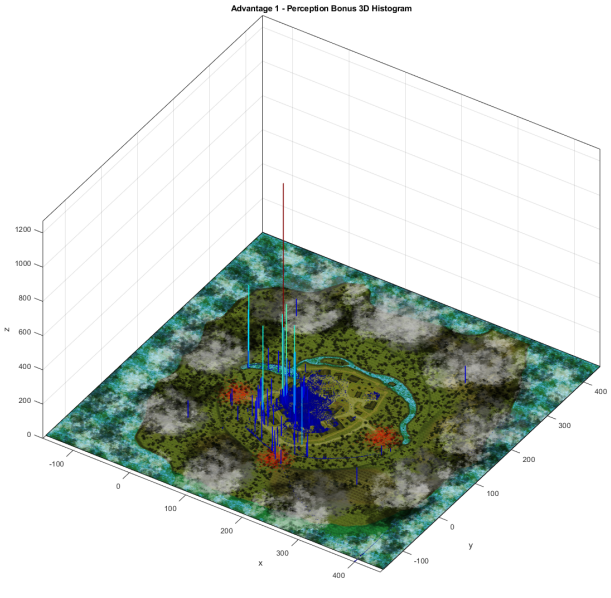
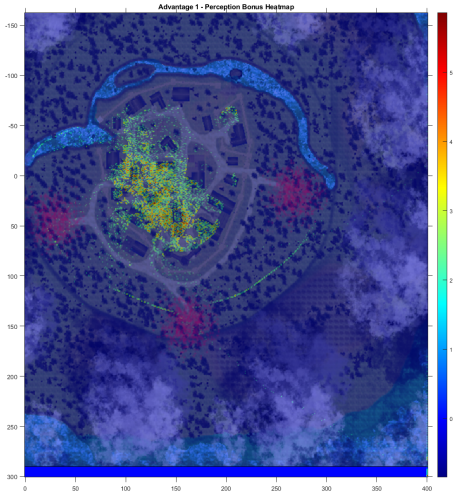
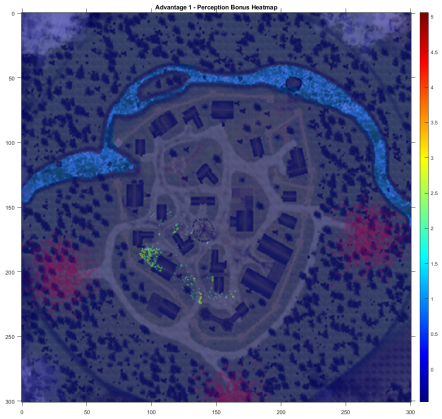


Figure 38: 3D Histogram of Advantage 1 Agents' positional Data

Outliers were identified and removed. These were commonly areas where the agents got stuck on the corner of obstacles, or mistakenly spawned inside objects. In Figure 39a a heatmap for the Agents with Advantage 1 can be seen. The primary activity of the agents takes place in the southern, western, and central part of the village, with some agent exiting through the village gates. Figure 39b displays a heatmap for the same agent population, but also for when agents had more than two neighbours. Here a small cluster of activity can be seen along the southwestern wall, outside the southern gate, and west of the HeartTree, with occasional groups scattered around.



(a) Heatmap of positional data with outliers removed for Advantage 1 Agents



(b) Heatmap of positional data with outliers removed for Advantage 1 Agents with 2+ nearby allies

Figure 39: Positional Data of Advantage 1 Agents

6.1.4 Advantage 2 - Pack Tactics - NPC Agent Heatmap Test Results

Data was collected for the positional data for the 10 Friendly NPC agents over a period of 14 hours. Of these 10, 3 of them were outfitted with the *Pack Tactics* advantage. This advantage provides a 100% damage buff to all nearby allies. Figure 40 shows a 3D histogram of the agents' positions, with additional bins representing the corners of the map.

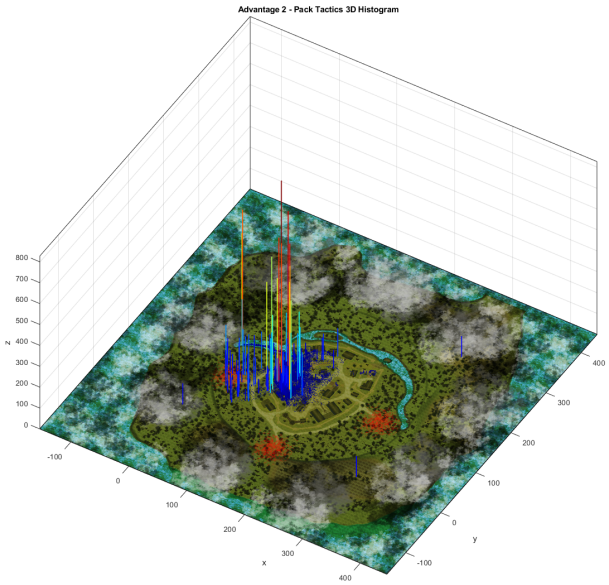
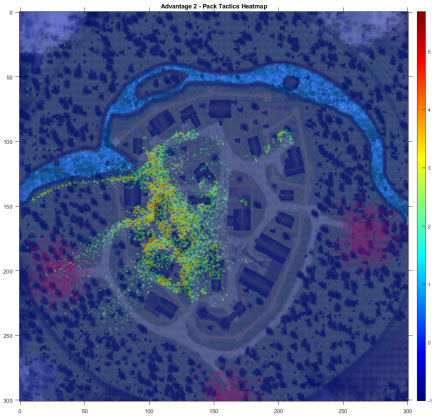
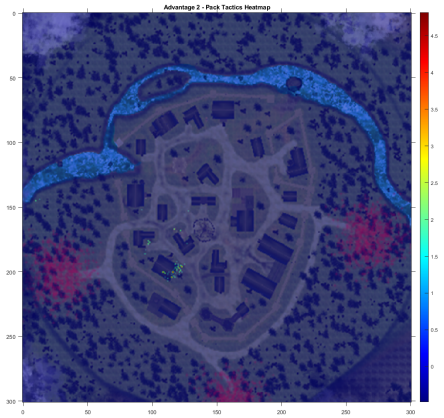


Figure 40: 3D Histogram of Advantage 2 Agents' positional Data

Outliers were identified and removed. These were commonly areas where the agents got stuck on the corner of obstacles, or mistakenly spawned inside objects. In Figure 41a a heatmap for the Agents with Advantage 2 can be seen. The primary activity of the agents takes place in the western and central part of the village, with agents exiting through the village gates to the west, along the river, or what appears to be over the wall. In Figure 41b a heatmap of the same population of agents can be seen. The primary activity here is beside one of the houses along the southwestern wall.



(a) Heatmap of positional data with outliers removed for Advantage 2 Agents



(b) Heatmap of positional data with outliers removed for Advantage 2 Agents with 2+ nearby allies

Figure 41: Positional Data of Advantage 2 Agents

6.2 Score Test

The score test is used to assess how effectively the agents completed their main and sub-goals. The way the score is calculated is presented in Figure 23.
The agent group that fought alongside agents with *Advantage 2 - Pack Tactics* obtained a considerably higher score compared to the agent group with *Advantage 1 - Perception Bonus*. A conjecture for this difference will be covered in Analysis of Results 7. The FSM group scores noticeably higher than both of the agent groups train through reinforcement learning with a score of 846.3749.

Score Table

FSM	Advantage 1 - Perception Bonus	Advantage 2 - Pack Tactics
846.3749	-5.1741	16.8173

Figure 42: Score Table for each Friendly NPC agent model

7 Analysis of Results

This section will cover an analysis of the results presented in Results 6, and compare the results from the different types of Friendly NPC agents.

7.1 Heatmap Analysis

7.1.1 Heatmap of Monster Heatmap

The Heatmap presented in Figure 35 shows a tendency for the agents to have been in the area near the HeartTree (marked with a purple circle). As the agents do not go directly towards the HeartTree when they spawn, and use time to walk around the environment, it can be very difficult to predict their next action. They spend a large portion of their time flailing around the environment attacking everything, and the resulting behaviour is very chaotic and unpredictable. This could be a function of the penalizing settings if the attacking mode function was set too low, in relation to striking the target. Another possibility is that the agents have not trained long enough.

7.1.2 Heatmap of FSM NPC Heatmap

The heatmap seen in Figure 37a displays the positional activity of the agents using the FSM model. As the agents are randomly assigned a new patrol point when their current target destination is reached, no agents wander off from their path unless an enemy enters their vision sensor. The area with the highest intensity is in the centre of the city where the HeartTree is located. I infer that this is due to the patrol point located next to the tree that the agents attempt to reach by using their NavigationMesh, in addition to the fact that this is the area with the highest monster activity, resulting in the FSM agents chasing the monsters to the tree when they spot one.

7.1.3 Heatmap of Advantage 1 - Friendly NPC Heatmap

In Figure 39a we can see that agents cluster around the western, southern and central parts of the village. The central activity aligns with their main-goal of protecting the HeartTree, while the western and southern activity is less clear. I speculate that agents learned that there is a lower chance of colliding with obstacles outside the village perimeter, and are actively looking for the exits that allow them to leave the village. This could also be caused by a confusion of their observation space, as their Raycast component sensors change lengths if they are near an ally with the *Perception Bonus* advantage. We can also observe two agents that successfully avoided the death-plane and then walked completely off the game world. This is likely a result of an oversight in the implementation, and is unexpected as the agents are rewarded based on how close they are to the HeartTree. The reward function for the distance to the HeartTree is an exponential function, approaching zero when far away, and quickly growing as the agent gets closer to the HeartTree. A cubic function that penalizes the agent when they go further away, and rewards them when they get closer could potentially have fixed this issue.
We also see that the northeastern parts of the village show limited activity. This could be due to the fact that while training, the model experienced frequent punishments in that area, and learned to avoid it.

7.1.4 Heatmap of Advantage 2 - Friendly NPC Heatmap

The heatmap seen in Figure 41a shows its primary activity in the western and central part of the village. We see a larger tendency to leave the village compared to the NPCs with Advantage 1, but only through the left gate and what appears to be over the wall. As the agents' reward score is based on the damage they deal to monster agents, high activity around the central area of the village where monster agents are shown to cluster, is an ideal area to hunt enemies. However, the activity seen on Figure 41b does not suggest that this behaviour is due to the advantage *Pack Tactics*, as the primary activity clusters for groups of 3 or more agents is further south. In a similar manner to the Advantage 1 group, this group shows no interest in the eastern side of the village.

7.1.5 Score Test

The FSM agents dominated the score test unsurprisingly. As the FSM agents uses a Navigation Mesh, they directly avoid all obstacles and are thus never penalized from this. Longer training times for the advantage groups could potentially have closed this gab significantly as their ability to avoid obstacles increases. The agents with Advantage 2 scores noticeably higher than the agents with Advantage 1. With both agents being rewarded based on the damage they deal to enemies and frequent they die, the damage boost provided by *Pack Tactics* plays a dominant role in their final score. The difference however is surprising considering how infrequent the Advantage 2 agents formed groups.

7.2 Trail Results

The ability to leave a trail behind for other agents to observe was applied to both agents, and while its direct influence was not measured, I believe it played a part in congregating the agents to areas other agents had visited.
With two exceptions in the group with Advantage 1 - *Perception Bonus*, the agents did not wander too far from where other agents had been. I infer that the agents could have learned that areas with few trails have a lower likelihood to provide high rewards, similarly to how the *Ant Colonisation* algorithm can work. Given enough training, I speculate whether the agents would generate a higher level of swarm-like behaviour.

7.2.1 Analysis of Results Conclusion

The heatmaps presented in Figure 39a and Figure 41a show very similar areas of activity. Primary activity is in the western and central parts of the village. Both agents show signs of wandering off from the main goal. This could have been caused by a number of elements and will be discussed in Design 8.1. The advantage agent scored significantly lower than the FSM agent, which avoids nearly all penalties from colliding with obstacles using its NavigationMesh. However, the advantage agents erratic movement behaviour covers a much larger area.
While it appears that there is a small difference in the behaviour of the agents with advantage 1 - *Perception Bonus* and the agents with advantage 2 - *Pack Tactics*, the data is insufficient to determine the significance of this. Test 8.2 will cover how the test could have been improved

8 Discussion

Centred around *Artificial Intelligence* of NPCs in video games, different types of artificial intelligence design methodologies and NPCs in State of the art games was researched, see Artificial Intelligence in games 2.1 and State of the Art Conclusion 2.3.6. To this end a prototype was developed with the aim to discover, observe and perhaps develop new and interesting behaviour through the use of indirect advantages and communication between agents. This section will focus on a discussion of the project as a whole, alongside a personal view of the direction I believe artificial intelligence can take within the video game industry.

8.1 Design

Game World
The observational space of the agents for this project was quite large. The agents are required to learn how to navigate around the environment, identify their surroundings and complete their individual main goals: *Attack / Defend* the HeartTree. I could have presented a more minimalistic art-style with less emphasis on the visuals of the game experience. However I decided to create as high a visual quality within the time constraints that I had. This modus operandi augmented my motivation and was at times a welcome break when needed.

Game Experience & Game Design
As covered in Gameplay Engagement 4.2.5, there are many features that combine to create an engaging experience. The core-game loop was only fulfilled to a limited degree and could have been developed more. While the original vision of the game was directed towards a combat RPG, time constraints meant that some of the core features that define the genre are absent. The difficulty of the game increases gradually over time as the monster's spawning rate increases as the game progresses. The player is outfitted with the ability to level up through a simple system that allows them to raise their health and stamina, as well as unlock the advantages the Friendly NPCs are trained with. The progression system for both the player and the game world is still lacking. The combat itself is quite basic, only featuring one type of weapon option. A choice between light and heavy attacks could have provided more variety concurrent with the game's progression.
The game contains some simple enemy variation, as the hostile NPC can take on the model of one of four different creatures: *Ghoul*, *Giant Rat*, *Evil Watcher* and *Oak Tree Ent*. While there is not any difference in their behaviour, as they each use the same trained model, they differ in terms of health and damage dealt. Enemy variety was implemented to make the experience feel less repetitive.

Level Design
The level that the agents are trained in presents the agents with a variety of different smaller obstacles, such as environmental props and larger obstacles such as the houses and village walls. From a human player's standpoint, the world is designed to be easy to navigate, with clear pathways, and unique landmarks such as the HeartTree in the centre, the Gallows, Well, and the Market area. Smaller obscure dirt pathways have been painted on the terrain to incentives a player's interest and lead them into back-alleys and behind houses where environmental props are positioned to tell a story about the inhabitants of the village.

Agents
The agent's characteristics could be improved in a variety of different ways. This could be by increasing their action spaces, training goal-dependent behaviour or changing the behaviour of the AI to maximize the game-flow for player experience, as was mentioned in State of the Art Conclusion 2.3.6. Goal dependent behaviour could potentially be achieved through more targeted curriculum learning, providing an environment with the desired goal in mind. By combining behaviour trees with reinforcement learning, one could allow agents to switch between different learned behavioural states when deemed necessary.
As the Monster agents and the Friendly NPC agents have an adversarial relationship, training the agent models in parallel could have been considered. Another approach could have been to use imitation learning, to create a baseline agent, the other agents could train against.
The environment was designed for the agents to be around group level with minuscule difference in elevation in the terrain. The agents have been observed to use small changes in elevation such as stairs or tree roots as ramps to leap up into the air before gravity brings them down. This is highly unlikely to be a planned action from the agents, but could explain how the agents occasionally appear to be passing through walls or other obstacles on the heatmaps, see subsection 11.4.
To speed up training time, the Friendly NPC agent's action of placing barricades to block paths and function as obstacles was removed. This ability would have provided the agents with a more versatile action space, and could potentially have resulted in new behavioural patterns emerging.

Game as an RPG
While the original vision of the game was directed towards a combat RPG, time constraints meant many of the core features defining the genre were absent. The combat itself is very basic, only featuring one type of weapon, and no choice between light and heavy attacks and providing no meaningful variety in the combat as the game progresses. One of the core features of main RPGs is the option of player choice, whether this is in the form of play style, advancement direction through level up, or story choices. The prototype for this project does not fulfil these features.

World Map
The world map created was exceptionally useful with regard to debugging the agents. With its use, one is able to get a sense of where all the agents are located at a given time, what their actions are and where they tend to move. As each agent has been provided with a randomly chosen first and last name on initialisation and thus keeping track of their progress was easier. The world map can be seen in Figure 43, and displayed the following information in real-time:

Randomly Generated NPC Agent Name, - NPC Agent Colour, - NPC Last Action that provided a reward (Including negative rewards), - NPC Reward value of Last Action, - NPC Pinned Location on overhead map (Colour Coded), - Pinned location of all Monsters on overhead map

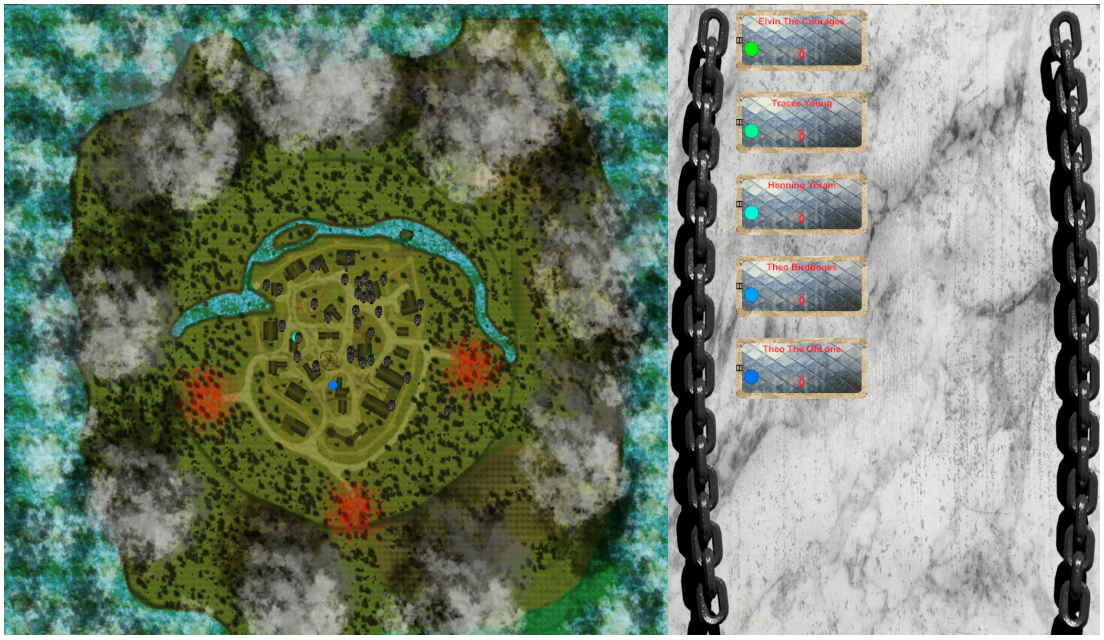


Figure 43: World Map

8.2 Test

Heatmaps
The heatmaps used for this project to help analyze the positional activity of the different agent were very effective and produced a lot of useful information. They gave not only a snapshot of the behaviour but also an overall conceptualisation of their behaviour as a whole. The agents were designed to maximize their score by completing their main goals, which lead to a certain degree of unpredictability as a single snapshot of their positional activity could vary a great deal. Another iteration on the heatmap methodology presented was considered which would show the significance of their position in relation to how far the agent was into a given episode. The present heatmap methodology does not take this into account, cloaking whether the agents spawn and immediately begin to navigate towards their main goal.

Score Test
Due to the difference in how the FSM agents and RL agents operate, the score comparison between the two AI types can be disregarded without finalizing the training model of the RL agents. The finish version of the advantage model reach an entropy level of 2.965(Advantage1) and 3.092(Advantage 2) whilst the Monster Agent achieves a level of 1.818. From this I infer that a prolonged training time could have resulted in a high score for the agents.

Between group experimental design test
A qualitative interview of participants playing through the prototype on the team of the Friendly NPC agents was considered, but not fulfilled due to time constraints. The aim of this interview would have been to get participants to describe the behaviour of their allied NPCs in a between group experimental design. Group A would have played alongside the Finite State Machine Agents, while Group B would have played alongside the NPC agents trained with Advantages. This approach was based on the hypothesis that more complex behaviour would require a more lengthy description than a simple behaviour.
This test could also be used to determine the player experience, and how they felt playing through the game when allied with either of the two groups.

8.3 Artificial Intelligence in Games

While I am excited about where the future of machine learning will lead the development of NPC agents, I do not however believe that they will completely replace traditional AI designs used in State of the Art games. Artificial agents in games need a specially tailored balanced approach depending on the vision of the developers. Agents are generally designed to be fair to the player and provide a fun experience. *Horizon Zero Dawn* is a fine example of this, creating breaks between machine attacks and not overrunning the player with every single machine nearby.
Mihaly Csikszentmihalyi flow theory encompasses a balance between player skill level and difficulty. As the player gets more proficient at the game there is a similar increase in difficulty. This maintains the *Flow*, interest, challenge and fun of playing. I propose that a similar relationship exist between agent complexity, and the players ability to learn and predict their behaviour. If every NPC behaved in a completely unpredictable manner, the player would then have no notable patterns to learn, and as a result the fun and game experience would be diminished. In contrast, too simple or too easily learned patterns would create games without challenges and surprises and bore the player. Being able to predict what the games NPCs will do is a requirement for many games of the *Stealth* genre, as the player often needs to learn the routes and behaviour of NPCs to be able to plan their approach and when to cause distraction. The AI must not be too clever, but rather develop challenges parallel to the player's ability; *Alien Isolation* is a great example of this as the *Xenomorph* unlocks branches of its behaviour tree over the course of the game as the player progresses the story.
I believe machine learning could potentially be used to help make less vital agents feel more realistic, while the core AI agents will still be designed through systems that allow game designers to have better control of the core behaviour. Based on the results presented in this project, I speculate that machine learning might be used to increase the complexity of behaviour trees, the conditions, transitions and the different behavioural states agents can find themselves in. I also speculate that machine learning can be used to assist in designing more challenging enemy AI, with the potential to assist in designing more intelligent agents.

9 Conclusion

This paper seeks to explore and observe whether artificial intelligence agents outfitted with indirect advantages and communication methods could develop new and interesting behavioural patterns compared to traditional State of the Art Ad-Hoc design methodologies.
Based on research into *Artificial Intelligence*, *State of the Art* games, and *Behavioural Psychology*, the following hypothesis was constructed:

Null Hypothesis: Agents when provided with indirect cooperative advantages will act predictably.
Alternative Hypothesis: Agents when provided with indirect cooperative advantages will behave less predictably and in a more diverse manner.

A game prototype was developed in *Unity*'s game engine, and two agent types with an adversarial relationship were designed to play the game: *Monster Agents* and *Friendly NPC Agents*. The *Friendly NPC Agent* was further divided into three groups: *Finite State Model*, *Reinforcement Learning: Perception Bonus Advantage*, and *Reinforcement Learning: Pack Tactics*. The two *Friendly NPC Agents* were trained against the *Monster Agents*, and finally, each *Friendly NPC Agent* ran for a 12 hour session collecting data on their positions and their game scores.
The results from the score test show that the Ad-Hoc *Finite State Machine* model significantly outperforms the *Reinforcement Learning* models in terms of completing the game's objectives. The results from the *Heatmap Test* displays chaotic and unpredictable behaviour from the *Reinforcement Learning* models. However, due to the limited training time of the agents, it is not possible to measure the exact impact that the indirect advantages have on the model, and the alternative hypothesis of: *Agents when provided with indirect cooperative advantages will behave less predictably and in a more diverse manner* can thus not be accepted.
Artificial Intelligence is a great tool for providing an exhilarating experience for the player. I believe that finding the the balance between AI predictability and complexity in the behaviour of the NPCs, is a possible solution to this. There should be room for the players to learn and for the NPCs the ability to surprise players as well. I believe *Machine Learning* can have a prominent part to play in the future development of NPCs in video games. Through the use of machine learning, it might be possible for enemy AI to learn individual players behaviours and habits, and consequently adapt their behaviour accordingly. This could create a more immersive, realistic and more challenging gaming experience by providing a precisely tailored game for the player, because the enemy has the learning capability to adapt to the player's individual preferences, ability and gaming style.

10 Bibliography

References

Fullerton, Tracy (Jan. 2018). *Game Design Workshop. A Playcentric Approach to Creating Innovative Games: 4th edition*, pp. 57–95. ISBN: 9780240809748. DOI: 10.1201/b22309.

Lara-Cabrera, Raul et al. (Jan. 2015). "Game artificial intelligence: Challenges for the scientific community". In: *CEUR Workshop Proceedings* 1394, pp. 1–12.

Yannakakis, Georgios N. and Julian Togelius (2018a). *Artificial Intelligence and Games*. 1st. Springer Publishing Company, Incorporated, pp. 32–33. ISBN: 3319635182.

– (2018b). *Artificial Intelligence and Games*. 1st. Springer Publishing Company, Incorporated, pp. 34–36. ISBN: 3319635182.

– (2018c). *Artificial Intelligence and Games*. 1st. Springer Publishing Company, Incorporated, pp. 37–39. ISBN: 3319635182.

– (2018d). *Artificial Intelligence and Games*. 1st. Springer Publishing Company, Incorporated, pp. 59–65. ISBN: 3319635182.

– (2018e). *Artificial Intelligence and Games*. 1st. Springer Publishing Company, Incorporated, pp. 71–76. ISBN: 3319635182.

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN: 0262039249.

OpenAI (2020). "OpenAI, Spinning Up, Proximal Policy Optimization". In: URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html#id3.

Roessingh, Jan et al. (Oct. 2017). "Machine Learning Techniques for Autonomous Agents in Military Simulations-Multum in Parvo". In: DOI: 10.1109/SMC.2017.8123163.

Griffin, Joe and Ivan Tyrrell (2004). *A new approach to emotional health and clear thinking*. HG Publishing, pp. 93–94. ISBN: ISBN 1 899398 31 7.

Donaldson, Alex (2017). "Monster Hunter World interview: "We've set up rules for how they should behave, but we didn't script monster behaviour"". In: Accessed 28/04/22. URL: https://www.vg247.com/monster-hunter-world-interview-weve-set-up-rules-for-how-they-should-behave-but-we-didnt-script-monster-behaviour.

Thompson, Tommy (2019). "Behind The AI of Horizon Zero Dawn (Part 1)". In: Accessed 28/04/22. URL: https://www.gamedeveloper.com/design/behind-the-ai-of-horizon-zero-dawn-part-1-.

Gilbert, Ben (2017). "One of the best PlayStation 4 games is just $20 right now". In: Accessed 28/04/22. URL: https://www.businessinsider.com/horizon-zero-dawn-ps4-photos-2017-11?r=US&IR=T#the-results-are-explosive-and-satisfying-18.

Dan Curtis, others (2018). "Undead Parish". In: Accessed 29/04/22. URL: https://www.ign.com/wikis/dark-souls/Undead_Parish.

Corden, Jez (2022). "Elden Ring guide: Melee combat, stances, poise, blocks, explained". In: Accessed 29/04/22. URL: https://www.windowscentral.com/elden-ring-melee-combat-guide.

Greenwood-Ericksen, Adams (2010). "Why Left 4 Dead Works". In: Accessed 28/04/22. URL: https://www.gamedeveloper.com/design/why-i-left-4-dead-i-works.

mixtape_maker (2019). "Left 4 Dead - Review". In: Accessed 28/04/22. URL: https://gamehag.com/news/left-4-dead---review1.

AIandGames, Tommy Thompson (2020). "Revisiting the AI of Alien: Isolation". In: Accessed 28/04/22. URL: https://www.aiandgames.com/2020/05/20/revisiting-alien-isolation/.

Rodriguez, David (2014). “Alien: Isolation Developers Describe Gameplay as "Unpredictable"”. In: Accessed 28/04/22. URL: https://www.dualshockers.com/alien-isolation-developers-describe-gameplay-as-unpredictable/.

Picard, R. W. (1995). “Affective Computing”. In: Accessed 03/03/2022.

Abbasi, Amir, Ding Hooi Ting, and Helmut Hlavacs (Jan. 2017). “Engagement in Games: Developing an Instrument to Measure Consumer Videogame Engagement and Its Validation”. In: *International Journal of Computer Games Technology* 2017, pp. 1–10. DOI: 10.1155/2017/7363925.

Schønau-Fog, Henrik and Thomas Bjørner (2012). “Sure, I Would Like to Continue: A Method for Mapping the Experience of Engagement in Video Games”. In: *Bulletin of Science, Technology & Society* 32.5, pp. 405–412. DOI: 10.1177/0270467612469068. URL: https://doi.org/10.1177/0270467612469068.

Przybylski, Andrew K., C. Scott Rigby, and Richard M. Ryan (2010). “A Motivational Model of Video Game Engagement”. In: *Review of General Psychology* 14.2, pp. 154–166. DOI: 10.1037/a0019440. eprint: https://doi.org/10.1037/a0019440. URL: https://doi.org/10.1037/a0019440.

Elion, Chris (2019). “Learning-Environment-Design-Agents.md”. In: Accessed 10/05/22. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_12_docs/docs/Learning-Environment-Design-Agents.md#isensor-interface-and-sensorcomponents.

11 Appendix

11.1 Appendix: A - MatLab Data Handling Script

```
fileName = 'AssetsSaveFile1.json'; % filename in JSON extension
fid = fopen(fileName); % Opening the file
raw = fread(fid,inf); % Reading the contents
str = char(raw'); % Transformation
fclose(fid); % Closing the file
data = jsondecode(str); % Using the jsondecode function to parse JSON from string


Image on Figure
map = imread('IslandMap.png');
imshow(map);

Getting positions for latest data

CheckedIds = [];
CollectedPositions = zeros(0, 2);
for i = length(data.SavedDataList):-1:1
    tempId = data.SavedDataList(i).UniqueID;
    id = str2num(erase(tempId,"Monster "));
    if(ismember(id, CheckedIds))
    else
        CheckedIds(end+1) = id;
        [tempX tempY] = getPositions(i, data);
        CollectedPositions = [CollectedPositions; [tempX tempY]];
    end
end

heatMapCornerLimits = 100;
for i = 1:10
    CollectedPositions = [CollectedPositions; [heatMapCornerLimits, heatMapCornerLimits]; ...
    [-heatMapCornerLimits, heatMapCornerLimits]; ...
    [-heatMapCornerLimits, -heatMapCornerLimits]; [heatMapCornerLimits, -heatMapCornerLimits]];
end
CollectedPositions = [CollectedPositions; [heatMapCornerLimits, heatMapCornerLimits]; ...
    [-heatMapCornerLimits, heatMapCornerLimits]; ...
    [-heatMapCornerLimits, -heatMapCornerLimits]; [heatMapCornerLimits, -heatMapCornerLimits]];
for i = 1:2

    CollectedPositions(:,i) = CollectedPositions(:,i) + heatMapCornerLimits;
end

heatX = CollectedPositions(1:end, 1);
heatZ = CollectedPositions(1:end, 2);
```

Figure 44: Matlab Data Handling Code Snippet 1

```
histo = histogram2(heatX, heatZ, 1000, 'BinMethod', 'integers','FaceColor','flat', 'FaceLighting', ...
    "flat", 'Normalization',"countdensity");
counts = histo.Values;

counts(counts>20) = 0;
highestZ = max(max(counts));
hold on;      % Add to the plot
xlabel('x');
ylabel('y');
zlabel('z');
img = imread('IslandMap.png');     % Load a sample image
imageSize = 300;
imageScaling = 2;
imageXOffset = -204;
imageYOffset = -200;

xImage = [0 imageSize; 0 imageSize] * imageScaling...
        +[imageXOffset imageXOffset; imageXOffset imageXOffset];   % The x data for the image corners
yImage = [imageSize imageSize; 0 0] * imageScaling...
        +[imageYOffset imageYOffset; imageYOffset imageYOffset];% The y data for the image corners
zImage = [0 0; 0 0];   % The z data for the image corners
surf(xImage,yImage,zImage,...    % Plot the surface
     'CData',img,...
     'FaceColor','texturemap');

outliers = counts > 5;
counts(outliers) = 0;
hold off;
G = counts;
% G(G ==0) = NaN;
G = rot90(G);
heatMap = imresize(G, [1000, 1000]);

heatMapObject = imshow(heatMap, [], 'XData', histo.XBinEdges, 'YData', histo.YBinEdges);
heatMapImageScaling = 1.7;
heatMapXOffset = -160;
heatMapYOffset = -170;
axis on;
colormap((jet(256)));
colorbar;
hold on
xImage = [0 imageSize; 0 imageSize] * heatMapImageScaling...
        +[heatMapXOffset heatMapXOffset; heatMapXOffset heatMapXOffset];   % The x data for the image corners
yImage = [0 0; imageSize imageSize] * heatMapImageScaling...
        +[heatMapYOffset heatMapYOffset; heatMapYOffset heatMapYOffset];% The y data for the image corners
zImage = [0 0; 0 0];   % The z data for the image corners
surf(xImage,yImage,zImage,...    % Plot the surface
     'CData',img,...
     'FaceColor','texturemap', "FaceAlpha",0.6);
hold off
grid off
```

Figure 45: Matlab Data Handling Code Snippet 2

```
Position Function
function [positionX, positionY] = getPositions(NPCNumber, dataList)

for i = 1:length(dataList.SavedDataList(NPCNumber).NPC_Position)
   a = cell2mat(dataList.SavedDataList(NPCNumber).NPC_Position(i));
   dataSplit = strsplit(a);
   npcX(i) = dataSplit(1);
   npcZ(i) = dataSplit(3);

    xFixingStructure = cell2mat(regexp(cell2mat(npcX(i)), '(?<=\().*?(?=\,)', 'match'));
    npcZ(i) = append(',', npcZ(i));
    zFixingStructure = cell2mat(regexp(cell2mat(npcZ(i)), '(?<=\,).*?(?=\))', 'match'));

   xRounded(i)= round(str2double(xFixingStructure),0,"decimals");
   zRounded(i)= round(str2double(zFixingStructure),0,"decimals");
end

roundedNpcXTransposed = xRounded.';
roundedNpcZTransposed = zRounded.';
npcXTransposed = npcX.';
npcZTransposed = npcZ.';
positionX = [roundedNpcXTransposed(:)];
positionY = [roundedNpcZTransposed(:)];
end
```

Figure 46: Matlab Data Handling Code Snippet 3

11.2 Appendix: B - Multi Purpose Liquid Shader

```
    {
        _MainTex("Base (RGB) Trans (A)", 2D) = "white" {}
        _Color("Tint", Color) = (1,1,1,0.4)
        _FresnelColor("Fresnel Color", Color) = (1,1,1,1)
        _FresnelBias("Fresnel Bias", Float) = 0
        _FresnelScale("Fresnel Scale", Float) = 1
        _FresnelPower("Fresnel Power", Float) = 1
        _BumpMap("Normal Map", 2D) = "Bump" {}
        _Shininess("Shininess", Float) = 10
        _SpecColor("Specual Material Color", Color) = (1, 1, 1, 1)
        _FlowSpeed("Flow Speed", float) = 1
        _MaxHeight("Displacement Height", float) = 1
            //_FlowDirection("Flor Direction", )
    }

        SubShader
        {
             Tags {"Queue" = "Transparent" "IgnoreProjector" = "True" "RenderType" = "Transparent"}
        ZWrite Off
        Blend SrcAlpha OneMinusSrcAlpha
        Cull front

            Pass
            {
                CGPROGRAM

        #pragma vertex vert
        #pragma fragment frag
        #pragma target 3.0

        #include "UnityCG.cginc"

        sampler2D _MainTex;
        float4 _MainTex_ST;
        fixed4 _Color;
        fixed4 _FresnelColor;
        fixed _FresnelBias;
        fixed _FresnelScale;
        fixed _FresnelPower;
        sampler2D _BumpMap;
        float4 _BumpMap_ST;
        float4 _SpecColor;
        float _Shininess;
        float _FlowSpeed;
        uniform float _MaxHeight;
```

Figure 47: Multi Purpose Liquid Shader, by Author

11.3 Appendix: C - Advantage Checker

```
1           if (nearbyAllies.Count != 0)
2           {
3               for (int i = 0; i < nearbyAllies.Count; i++)
4               {
5                   if (nearbyAllies[i].GetComponent<FriendlyNPCAgent>())
6                   {
7                       if (!nearbyAlliesPerception.Contains(nearbyAllies[i]) && ...
                            (nearbyAllies[i].GetComponent<FriendlyNPCAgent>().advantage1_PerceptionAdvantage))
8                       {
9                           nearbyAlliesPerception.Add(nearbyAllies[i]);
10                      }
11
12                      if (!nearbyAlliesPackTactics.Contains(nearbyAllies[i]) && ...
                            (nearbyAllies[i].GetComponent<FriendlyNPCAgent>().advantage2_packTactics))
13                      {
14                          nearbyAlliesPackTactics.Add(nearbyAllies[i]);
15                      }
16
17                      if (!nearbyAlliesAoEHealing.Contains(nearbyAllies[i]) && ...
                            (nearbyAllies[i].GetComponent<FriendlyNPCAgent>().advantage3_AoEHealing))
18                      {
19                          nearbyAlliesAoEHealing.Add(nearbyAllies[i]);
20                      }
21                  }
22                  else if (nearbyAllies[i].GetComponent<PlayerController>())
23                  {
24                      if (!nearbyAlliesPerception.Contains(nearbyAllies[i]) && ...
                            (nearbyAllies[i].GetComponent<PlayerController>().advantage2_PerceptionAdvantage))
25                      {
26                          nearbyAlliesPerception.Add(nearbyAllies[i]);
27                      }
28
29                      if (!nearbyAlliesPackTactics.Contains(nearbyAllies[i]) && ...
                            (nearbyAllies[i].GetComponent<PlayerController>().advantage1_packTactics))
30                      {
31                          nearbyAlliesPackTactics.Add(nearbyAllies[i]);
32                      }
33
34                      if (!nearbyAlliesAoEHealing.Contains(nearbyAllies[i]) && ...
                            (nearbyAllies[i].GetComponent<PlayerController>().advantage3_AoEHealing))
35                      {
36                          nearbyAlliesAoEHealing.Add(nearbyAllies[i]);
37                      }
38                  }
39              }
40              int tempPerceptionPassCounter = 0;
41              for (int j = 0; j < nearbyAlliesPerception.Count; j++)
42              {
43                  if (nearbyAlliesPerception[j])
44                  {
45                      tempPerceptionPassCounter++;
46                  }
47              }
48
49              int tempPackPassCounter = 0;
50              for (int j = 0; j < nearbyAlliesPackTactics.Count; j++)
51              {
52
53                  if (nearbyAlliesPackTactics[j])
54                  {
55                      tempPackPassCounter++;
56                  }
57              }
58
59              int tempAoEPassCounter = 0;
60              for (int j = 0; j < nearbyAlliesAoEHealing.Count; j++)
61              {
62
63                  if (nearbyAlliesAoEHealing[j])
64                  {
65                      tempAoEPassCounter++;
66                  }
67              }
68
69              nearbyAllyHasPerception = tempPerceptionPassCounter > 0 ? true : false;
70              nearbyAllyHasPackTactics = tempPackPassCounter > 0 ? true : false;
71              nearbyAllyHasAoEHealing = tempAoEPassCounter > 0 ? true : false;
72
73          }
74          else
75          {
76              nearbyAllyHasPerception = false;
77              nearbyAllyHasPackTactics = false;
78              nearbyAllyHasAoEHealing = false;
79              nearbyAlliesPerception.Clear();
80              nearbyAlliesPackTactics.Clear();
81              nearbyAlliesAoEHealing.Clear();
82          }
83
84          bool[] tempArray = new bool[3];
85          tempArray[0] = nearbyAllyHasPerception;
86          tempArray[1] = nearbyAllyHasPackTactics;
87          tempArray[2] = nearbyAllyHasAoEHealing;
88
89          return tempArray;
90      }
```


Figure 48: Code snippet - Does Nearby Allies have Advantages Function

11.4 Appendix: D - Friendly NPC Slope Exploit

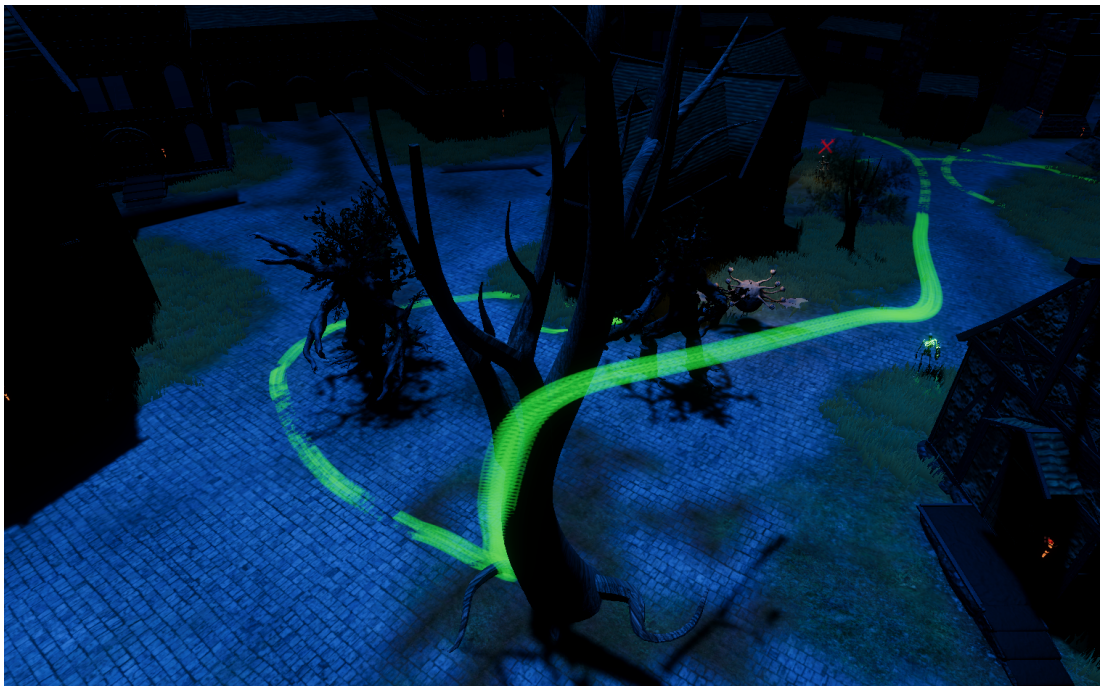


Figure 49: Friendly NPC Agent uses slope to gain altitude

11.5 Appendix: E - Unity Project

The entire unity project including all scripts used and the models and animations can be found in the attached folder: *Med10_Sebastian_Martin_Young_20173978_Project_Folder.zip*

11.6 Appendix: F - Playable Game

A playable build of the game is attached along with this thesis, and can be found in the folder: *Med10_Sebastian_Martin_Young_20173978_Project_Build.zip*