Student Name: Joshua Shepherd

Student Number: 1700471

Module Code: 6CS007

Module Name: Project and Professionalism

Project Title: A machine learning system for adapting game characters to human player behaviour

Module Leader Name: Dr. Rupert Simpson

Supervisor Name: Dr. Thomas Hartley

Reader Name: Dr. Mohammed Patwary

Submission Date: 10/05/2021

(The date you handed in the assessment)

Award Title: Computer Science (Games Development)

(Award Title for your project, if in doubt refer to your course/Module Registration)

School of Mathematics and Computer Science

Declaration Sheet

Presented in partial fulfilment of the assessment requirements for the above award.

This work or any part thereof has not previously been presented in any form to the University or to any other institutional body whether for assessment or for other purposes. Save for any express acknowledgements, references and/or bibliographies cited in the work. I confirm that the intellectual contents of the work are the result of my own efforts and of no other person.

It is acknowledged that the author of any project work shall own the copyright. However, by submitting such copyright work for assessment, the author grants to the University a perpetual royalty-free licence to do all or any of those things referred to in section 16(i) of the Copyright Designs and Patents Act 1988. (viz: to copy work; to issue copies to the public; to perform or show or play the work in public; to broadcast the work or to make an adaptation of the work).

Student Name (Print): Joshua Shepherd

Student Number: 1700471

Signature:  Date: 10/05/2021

(Must include the unedited statement above. Sign and date)

Please use an electronic signature (scan and insert)

How can Artificial Intelligence in games adapt to a human player?

By Josh Shepherd

Abstract

As video games proceed to push the boundaries of how the next generation of video games perform, game developers are searching for innovative ways to improve their artificial intelligence (AI) systems. AI has traditionally been designed and implemented with the ability of having different difficulty levels, while also using tried and tested techniques for implementation. While current techniques could potentially be sufficient and match a specific need, determining the right technique can be a difficult choice for a developer. This paper takes a look at the currently available techniques and how they have been implemented in the past, while also searching for a new combination of techniques that can create a unique and innovative system when combined. The paper then sets out to implement the combination in a resulting artefact and participate in testing the artefact to collect results from participants.

The full source code and the resulting artefact is open source online and can be accessed by navigating to github.com/JoshLmao/6CS007-ProjectBoss.

Contents

Abstract 3
1. Introduction 5
2. Literature Review of Adaptive Artificial Intelligence in Video Games 6
2.1 What types of Video Games AI exist? 6
2.2 Impact of Artificial Intelligence in Games 7
2.3 Artificial Intelligence Techniques Used in Games 8
2.3.1 Finite State Machines 9
2.3.2 Goal-Oriented Action Planning 10
2.4 Machine Learning in Video Games 11
2.5 Conclusion 12
3. Designing a system to adapt to a human player 13
3.1 Introduction 13
3.2 Goal Oriented Action Planning (GOAP) 14
3.3 Machine Learning 15
3.3.1 Random Range Machine Learning Model 16
3.4 Implementation Tools 17
4 Implementation of an adaptable virtual character 18
4.1 Goal-Oriented Action Planning (GOAP) 18
4.2 Machine Learning in Python & Unreal Engine 20
4.3 Logging inside Unreal Engine 22
5. Experimentation & Testing 23
5.1 Testing with Human Players 23
5.2 Questionnaire Results 24
6. Conclusions 26
7. Evaluation of Product 27
7.1 Product Evaluation 27
7.2 Sources Evaluation 27
7.3 Self-Reflection 27
7.3.1 Overall Artefact and Results Reflection 27
7.3.2 GOAP Reflection 28
7.3.3 Machine Learning Reflection 28
8. Product Management 29
9. References 32

1. Introduction

Video games may have started as simple beginnings, but the industry has turned quickly into a multibillion-dollar industry. With the newest video games all competing for players, the latest and greatest games are required to step above and beyond what has previously been made. On area of video games that gets overlooked by players is what goes on within an artificial intelligent (AI) character, such as a non-playable character like an enemy or ally soldier. These characters make up a big section of a game since without them, the player would be alone in a world by themselves.

This report will focus on designing and creating the ability for a virtual character within a video game to be adaptable to a player. The report aims to look into the current systems available to developers to creating artificial intelligence, design and implement a system focused on adaptability inside of an artefact and evaluate the usage results of the artefact. With clear aims, the main objectives of the report are being able to implement an AI system on a virtual character, test the character on its adaptability, as well as validate the implementation results to see if the system can adapt to any player and their play styles.

What a video game is has changed a lot since the first ever was created in October 1958. In the past, games were inspired by existing physical games, such as chess, drafts, and tennis due to the hardware limitation at the time. Because of this, these games were usually simple sprites that could be moved, with some sound effects. However, as the technology improved, the scope of what could be contained within a game also grew. Now, developers are able to create their own world, fully customizable from sound, models, textures, to using sophisticated artificial intelligence systems to control virtual characters.

AI characters in a video game are important, but so is the technology and techniques behind them that make them a fun and competitive aspect of a game. An AI system should be able to make effective decisions that support the experience the developer aims to create (Rabin, Game AI Pro 2, 2015). As there are many systems and designs out currently available to developers, being able to utilize the best one for their use case can be quite a challenge.

Whilst the most basic systems are able to provide a framework for developers to get up and running with a simple AI system, they lack the complexity and adaptability that is required by modern players. If an AI system is not very interactive or is too “artificially stupid” (Rabin, Game AI Pro, 2013), the player will quickly learn the whole system and get bored. However, if an AI system is too complex and too big in size, players can feel overwhelmed and defeated by the system. Being able to find a good balance between these extremes, whilst also creating a system that is adaptable and fun is one of the main featured required in today’s video games.

This paper will discuss existing systems and techniques to determine how one such AI system and technique can be utilized and used hand in hand to create an enemy agent. The agent’s final performance will display how these two elements can work together to adapt to the player and provide and enjoyable experience.

2. Literature Review of Adaptive Artificial Intelligence in Video Games

Artificial Intelligence (AI) in video games has and will always be important and strive to improve the quality of the game (Bakkes, Spronck, & van den Herik, 2009). AI is one of the core building blocks to a video game and one of the main pillars to creating the next best game loved all over the world. Because of this, being able to choose, design and implement effective artificial intelligence systems becomes similar to a work of art, as a lot of time is spent on designing, implementing, and reworking these systems to get the most desired outcome for the system.

As Rabin states, the goal of game AI is similar to Disney's aim of creating artificial life through animation (Rabin, Game AI Pro, 2013). The aim of both of these mediums is to entertain the player/viewer into believing what they see, for them to become captivated into the game or film and to become emotionally attached to the characters. This connection between the player and the game is unique and being able to create that is a big part of what makes artificial intelligence great.

Over the last ten years, gaming has changed tremendously and has led to inexpensive and powerful hardware (Risto Miikkulainen, 2006). Since more powerful hardware for a lower cost has enabled more people to own the hardware, it enables more people to take the step into learning about games development and all of its aspects. As existing artificial intelligence techniques have been used for such a long time, new techniques from young and fresh developers to fulfil specific use cases can appear. With better hardware, developers have been able to create better systems that can perform more validation or more procedural checking before execution, as well as being able to run more computational-heavy processes during runtime. This also adds more scope that developers can add to their systems, making them greater at runtime and more effective at its main purpose.

When it comes to Artificial Intelligence, the last decade has seen many leaps in advancements, thanks to advances in the area of deep learning or deep neural networks (Ford, 2018). These go hand in hand with the advances in the hardware inside CPU's and GPU's, such as NVIDIA's GPUs containing more and more computing power year over year. With such progression enabling developers to perform more computations per frame, it allows for more complex algorithms or much more advanced math computations to take place.

2.1 What types of Video Games AI exist?

In video games, certain types of games will require the use of artificial intelligence to simulate virtual characters playing against a human player. Video games have many different genres such as first-person shooters, strategy games, platforms, stealth games and fighting games which require AI systems to be versatile and offer a wide feature set to create the best AI for that genre.

Artificial Intelligence inside video games contains a lot of complexity and potential for advanced methods. When you think of video game AI, you might jump to thinking of a shop keeper, or a simple town villager walking and having conversation with the player, however AI can be opened up to be much more complex. Micael makes the comparison that creating video games gives people a lot of passion as it allows us to create a world completely imagined by us, which is similar to playing god (DaGraça, 2017). With enough passion and effort, a single developer is able to fully

create their own world, completely hand crafted and tuned to their own desires. With this level of control over a world, developers also need the same level of control over AI systems.

When implementing an artificial intelligence system, being able to keep behavioural consistency is an important part of AI performance as it is also the 'presentation layer' of the AI (Sebastian Welsh, 2005). The outcome of the system should be able to maintain the illusion that it is a challenging fight for the player. The main issue with implementing systems for video games is that, like Micael (DaGraça, 2017) states, humans play differently every time which makes it a huge task for programmers to input all the data. As there are so many possibilities, choosing the best solution is a decision that requires extra thinking from the developer.

Choosing and designing an AI system that can best fit a task can be a hard job since there are a handful of systems to choose with varying ranges of complexity and outcomes. With a massive range of implementation methods available already that have already been used many times before, the choice of picking the best one for the situation can become difficult without knowing each implementation's advantages and disadvantages.

2.2 Impact of Artificial Intelligence in Games

When it comes to artificial intelligence (AI) in games, it should be about one thing and one thing only: being able to create a compelling experience for the player. (Rabin, Game AI Pro, 2013) As the focus of a game is the player, being able to adapt and fulfil their expectations can be difficult with such a wide range of players. Being able to create and manage agents created by the developers makes a situation where the game creators are able to manipulate and fine tune their AI to their own design, such as making an enemy that will only fight in a certain style.

Artificial Intelligence, in the world of computer science, can come in many different shapes and forms, however AI in video games is mostly thought as being an enemy or non-playable character (NPC) in the world. Mike Loukides, in a general sense, describes AI as being solely about building systems that answer questions and solve problems (Mike Loukides, 2018). In video games, the problem to solve can vary from being a simple NPC like a bartender, allowing the player to talk and trade, to being the complex final boss in skill-focused video game.

Since artificial intelligence is about managing and meeting the player's expectation, being able to avoid Artificial Stupidity is important to keeping the player's immersion. Artificial Stupidity is when the AI does something the player is not expecting such as being stuck on geometry or walking off a cliff (Rabin, Game AI Pro, 2013). Developers and designers of artificial intelligence need to iterate and test to make sure their designed artificial intelligence system does not fall prey to artificial stupidity and is able to achieve its aims.

Artificial intelligence, within the context of a video game, is about providing an entertainment value to the player. However, in the real world, the purpose of applying artificial intelligence is instead to act rationally or serve a desired purpose. As Stuart Russell (Russell, 2016) says, "all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment … adapt to change, and create and pursue goals". As a player's enjoyment is the goal, being able to manufacture a system that can provide fun, and entertainment is the primary objective to creating a fun and engaging video game agent.

2.3 Artificial Intelligence Techniques Used in Games

As there are many genres of video games, there are also many different types of artificial intelligence that can be applied and used. Different genres also contain many different potential characters that are required. From a shop keeper to an animal companion or an enemy soldier, the potential to create whichever character developers require is wide and vast. However, most characters usually fall into one of the many categories below, as stated by Mac Namee (Mac Namee, 2004):

- **Adversarial Opponents:** Usually take the form of an opponent used to fight the player. These opponents have a ranging of difficulty to make the game easier or harder for the player, but are most commonly used in single-player games
- **Strategic Opponents:** Used in games where the player/team takes control of a mass number of units and resources. Strategic opponents take a high-level view of the game and are able to take control of units and build formations and execute actions for those units (Mac Namee, Proactive Persistent Agents, 2004).
- **Partner Characters:** These are characters that are there to aid the character or story of the game, such as either by talking to the player or performing an event in the world. Partner characters are meant to be beneficial to the player and not be a detriment to gameplay and the player's progress.
- **Semi-Autonomous Units:** Characters that are able to be autonomous and make their own decisions, however the player can override and set their own actions. These are commonly used in real-time strategy or simulation games.
- **Support Characters:** Background characters that are not important to the main path of the game. These characters should not impact the player in any way and their main purpose can be to populate an area, such as shoppers in a marketplace or an audience in a stadium.

Since there are many types of artificial intelligence that have a wide range of functions, from being a challenge to the player or benefitting them, there are also many ways that developers can implement them. Each technique has their own advantages and disadvantages as can be seen in the overview table in figure 2.1.

AI Technique	Source	Advantages	Disadvantages
Behaviour Trees (BT)	(Neufeld, Mostaghim, Sancho-Pradel, & Brand, 2019)	Simple, easy to maintain and scalable. Can be used in combination with a planner, but not required.	Unable to have reactive behaviour. Usually used for simple, short behaviour plans.
Finite State Machines (FSM)	(Bourg & Seemann, 2004)	Simplistic implementation. Works well on small scale usage.	Does not scale well with many states.
Goal-Oriented Action Planning (GOAP)	(Orkin, Applying Goal-Oriented Action Planning to Games, 2008)	Defines an initial and goal world state. Determines best actions through a heuristic.	Maintaining of planner and many actions. Has a high implementation complexity.

			Much more computationally expensive.
Hierarchical Task Network (HTN)	(Georgievski & Aiello, 2015)	Similar to GOAP, defines initial and goal world state. Fast, scalable, and suitable for real-world usage. Breaks tasks up into smaller tasks.	Requires a well-structed knowledge of the system. Many different types of HTN systems. Most suitable for domains where a hierarchical representation is desirable
Dynamic Scripting	(Spronck, Ponsen, Sprinkhuizen-Kuyper, & Postma, 2005)	Generates a new script each time for every agent, based off of a rule base Easily to be understood for developers.	Not effective for specialized agents. Does not provide satisfying results against "expert" players

Figure 2.1: *Overview of AI techniques, with the advantages & disadvantages*

While these are just some of the most popular artificial intelligence techniques used in video games today, they all still contain their own unique challenges that occur either during implementation or at runtime. For example, let us take the Finite State Machines (FSM) and Goal-Oriented Action Planning (GOAP) AI techniques and take a deeper look into them.

2.3.1 Finite State Machines

One simple strategy that is quite common in AI is to use a Finite State Machine. As Bourg and Seemann describe (Bourg & Seemann, 2004) finite state machines date back to the early days of game programming. However, as they are simple to understand, implement and debug, they are still very common and frequently used in modern games development. The disadvantage of FSM's compared to the GOAP system is that actions are not planned in real time, with each action having transition requirements that require to be fulfilled. While finite state machines for basic behaviours are suitable, FSM's will have problems when developers try to scale up the solution as when the number of states and transitions increase, the complexity and visual readability does too. (Gill, 2004) Because of this, it is one of the reasons finite state machines are commonly used, from basic actors to an enemy actor in a game.

Finite State Machine's date back to early days of programming and are relatively easy to understand (Bourg & Seemann, 2004). They are implemented by using simple logic and only allow the state machine to be in one state at one time. They consist of a set of states, a set of inputs, a set of outputs and a transition (D. Johnson, 2001). States define what behaviour the enemy will run at that time. States also have transitions that define what requirements are needed to move from state to another. For example, a state of 'Patrolling' has behaviour for the enemy to move between one position and another, while the transition requirement would be the player has to be within 500 units. When the player is in range, the state would change to an 'Attacking' state.

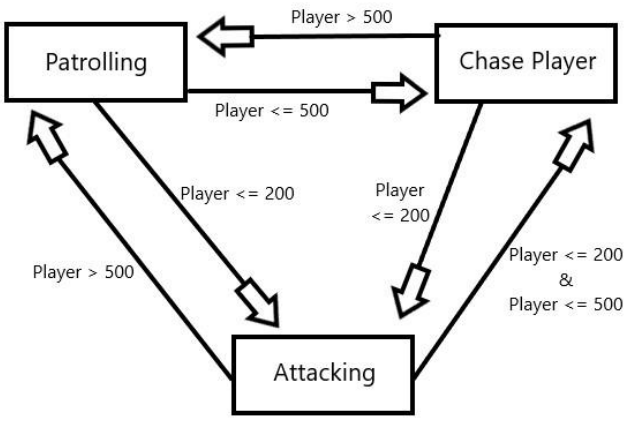


Figure 2.2: *An example of a basic finite state machine*

The most popular usage of a Finite State Machine in a video game is it is use inside the original Pac Man. In the original, the ghosts that chase the player use FSMs for their states which can be either roam freely and chasing or evading the player. David Bourg states that in this example, the ghost behaves differently depending on their state. If the player ate a power pill, then the ghosts evade state's requirements have been met which allow it to change (Bourg & Seemann, 2004). Finite State Machines also made an appearance in the highly popular Half Life 1, created in 1998 by Valve. As Alex states, even looking back at the implementation during that time, the code for Half Life contains a lot of simple and effective methods on implementing finite state machines (Champandard, 2017).

2.3.2 Goal-Oriented Action Planning

Another strategy to effectively manage artificial intelligence is to use a technique called Goal-Oriented Action Planning (GOAP) in combination with an effective planner to create a believable AI system. Jeff Orkin (Orkin, Applying Goal-Oriented Action Planning to Games, 2008) describes GOAP as a system where a character formulates his own plan to satisfy his goals. Because of this, the character exhibits less repetitive and predictable behaviour and can adapt his actions to fit the situation. Goal-Oriented Action Planning has already been designed and implemented in released video games such as *No One Lives Forever 2* and *F.E.A.R*, both created by Monolith Productions.

Goal-Oriented Action Planning is a system that is able to use a 'planner' to formulate a sequence of actions to satisfy a goal (Orkin, Applying Goal-Oriented Action Planning to Games, 2008). This system contains two main parts, a planner, and an array of actions, that combine to create a realistic and real-time AI system. To start creating a system using GOAP, developers and the GOAP system need to have an array of actions that make the character do something, like an action. For example, firing its weapon, reloading, diving to cover, regrouping with other AI, etc. Actions are required to have a precondition(s) and an effect(s). An action's effects are able to affect a tracked game state, such as reducing the players health, so as to reach a target state. The other part of the GOAP system is the use of a planner. The planner starts with a start state, considers all of the actions within the system available for that character, and works through all actions and their preconditions and effects, to formulate the best path to the goal state.

When designing their system for F.E.A.R, the developers wanted F.E.A.R to be an over-the-top movie experience with intense fights and choosing the GOAP system, they were able to provide a practical solution for that generation of games that was able empower AI characters with the ability to reason. (Orkin, Three States and a Plan: The A.I. of F.E.A.R., 2006) The final product of their system receives very high praise, as still does to this day, since the enemies react in real time and can communicate with squad mates to coordinate an attack on the player.

Figure 2.3 illustrates a simple overview of a goal-oriented action planning system. In the figure, it shows the start and goal state that the system uses to start and end with. However, in between those states can be any number of actions, represented in a circle, to help guide the system towards that goal state. Internally, the system considers each action's effects, such as reducing the agent's health or using its power, to determine the path towards its goal state.

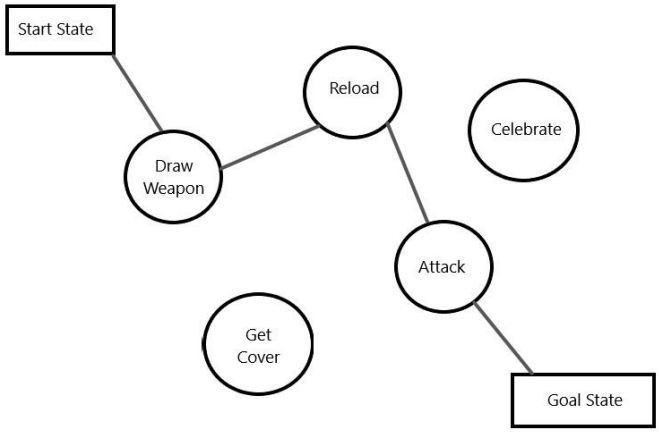


Figure 2.3: *Overview of how goal-oriented action planning formulates a plan from actions*

2.4 Machine Learning in Video Games

As there are many genres, types and systems that can be used within artificial intelligence, developers can also use techniques to help adapt their existing systems. Machine learning is one technique many developers utilize as it can produce life-like results. Machine Learning is described as being "A computer program [that is] said to learn from experience with respect to some class of tasks and performance measure…" (Zhang, Su, & Wang, 2007). As a machine learning system is able to use its past experiences to influence its future decisions, this can lead to innovative actions being taken by the system to perform tasks. With more experience, a machine learning system could be trained to evolve into an unstoppable enemy or an impressively accurate predictor inside of a video game environment.

Many applications of machine learning have already been used within a video game. For example, Bertens et al. (Bertens, Guitart, Chen, & Perianez, 2018) use machine learning as a way of recommending in-app purchasable items within a video game. With such a system, he was able to predict and display the most relevant in-app purchase for the player which led to an increase in profits. This application of machine learning is one method that games developers in recent years are using machine learning to influence players and increase profits from gathering data from its players.

However, a machine learning system can also be applied to control a character within a video game. Brown (Brown, et al., 2017) were able to use a machine learning system to act as a player in chess and trained it from previous games played by grandmasters. With the experience of grandmasters, their system "led to very high win rates (around 85%)" (Brown, et al., 2017). While if playing against this system you were highly likely to lose, it still shows how effective the previous experiences are at being able to defeat players.

With such a big amount of data being generated by players every second (Bertens, Guitart, Chen, & Perianez, 2018), being able to utilize and use that data to train a machine learning system to better enhance a video game is a unique and impressive technique that contains the potential to create a challenging rival enemy. While an undefeatable enemy is not fun for the player, being able to tune the system from undefeatable to a fun experience is an important tool of machine learning during implementation.

2.5 Conclusion

Overall, artificial intelligence has made a lot of progress over recent years and will continue to grow, iterate, and develop. In video games, there are a handful of standardised techniques that are very common practice across the industry for implementing game characters using artificial intelligence. Within video games, developers have a handful of methods available to them when implementing a system. While all these methods have different advantages and disadvantages, choosing the best system is up to the developers. While Goal-Oriented Action Planning (GOAP) can very quickly increase in complexity and become inflexible (Mitchell, n.d.), there are methods such as Finite State Machine's (FSM) that can be a good starting point when developers start creating their own system. However, GOAP is a system that has a proven track record, in video games like F.E.A.R, of having enemies that are challenging and much more realistic.

This project will explore adversarial AI in video games as they are one of the most common forms when thinking about game AI, as well as ranging in various skill forms to present a different level of a challenge to the player. Adversarial opponents are able to present interesting challenges to the player as an enemy can have creative action behaviours that the player will have to learn to beat, while also providing a core challenge to the experience. This project will also explore goal-oriented action planning. GOAP seems to be the most advantageous as it offers an intuitive planner and will determine the best plan for that moment. The system can also utilize machine learning to provide life-like results and will enhance the final result of the agent.

3. Designing a system to adapt to a human player

3.1 Introduction

When thinking of answering the question “How can Artificial Intelligence in games adapt to a human player?”, one of the main features of that question is the use of artificial intelligence. Artificial intelligence is super common in video games, almost to the point where almost every game that is released contains AI, in one form or another. AI is mostly found in the most common form of a non-playable characters (NPCs) such as a shop keeper or an enemy soldier, but it can also be used in methods such as calculating a player outcome model that is able to calculate what the player might do next. To create one of these forms, there are a handful or options and methods of implementing AI, all with their own advantages and disadvantages.

When I first investigated all of these different methods to implement artificial intelligence, there were two uncommon methods that I had not heard of before that sounded very interesting. The two I found are called goal-oriented action planning (GOAP) and hierarchical task network planning (HTN). Both have previous been implemented in released video games and both have received praise.

While both systems sounded intriguing to me, I ultimately chose to use GOAP as I have not had much experience with implementing artificial intelligence and wanted to use a system that was complex, but manageable within the given timeframe. Also, I would be able to implement a machine learning model that could take in the relevant data from the GOAP system and create a new “updated cost” for each action after a plan had finished executing.

Once I settled on goal-oriented action planning as my artificial intelligence system to implement, I began to think, design and plan how I wanted to implement the system, as well as how my machine learning model would take previous action data and train itself on that. I started out by researching and designing my GOAP system and then moving onto my machine learning system since the latter relies on GOAP being created and working. These were the only things I planned since they were the most complex and were new to me.

I knew that I want my artefact to be a third-person game, with simple controls and actions by the player, such as having a small array of abilities to perform with as little number of keys to press as possible. My characters were going to be based off of the game Paragon as Epic released all of their character models for free to use inside of Unreal Engine. These models also come with full sounds and animations which is a massive bonus for getting started.

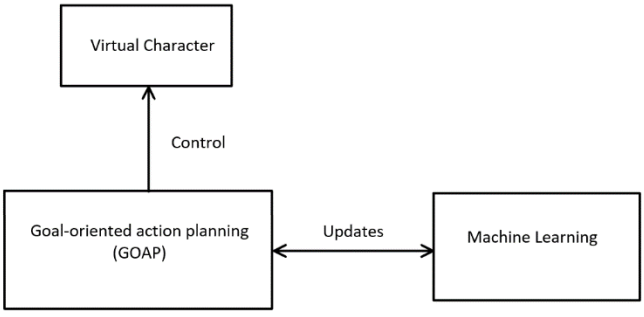


Figure 3.1: *High level overview of the designed system*

Figure 3.1 shows a basic overview of how each of the separate systems will work together to create an adaptable character. The goal-oriented action planning (GOAP) is at the centre of the system, where it communicates one way to only control the virtual character and perform actions. The machine learning part of the system is a two-way communication, where the GOAP system is able to update the machine learning's experience, as well as receive updates from the learning to better refine the GOAP's performance.

3.2 Goal Oriented Action Planning (GOAP)

Goal oriented action planning (GOAP) is a system for implementing artificial intelligence that uses a different structure to the more popular methods for games development. GOAP's main features are that the system uses a planner, most commonly a A* pathfinding algorithm, and a list of actions that are specified with preconditions, effects, a cost, and the functional code for that action.

As GOAP has previously been praised in the past in game such as F.E.A.R for being almost lifelike behaviour, I wanted to attempt at implementing such a system on one special enemy. The aim is that this one special enemy would be able to execute the most appropriate action, depending on the target world that is set.

World State Design

The world state is one of the two most important parts of the GOAP system. As such, I designed the world state in a simple way since this was my first time using the system and wanted to allow for an easy change, if required. The world state mostly contains flags that are related to the enemy's stance in the world, such as being in cover or within melee range. Apart from these, the world state only contains two flags that are used to reach an end action in the GOAP planner that will create an action sequence. The overall world state contains the following flags with a description of the purpose of that flag.

Project Boss World State

- "**in-cover**": Is the enemy currently in a cover, protected or out of sight of player.
- "**in-melee-range**": Is the enemy within very close distance to the player.
- "**is-invisible**": Is the enemy invisible to the player or its surroundings.
- "**heal**": Target state, used to self-heal the enemy.
- "**damage-player**": Target state, used to deal damage to the player.

The flags are heavily related to the following designed actions, since the two were designed in tandem.

Actions Design

The actions I designed for the boss were initially designed around the character model I wished to use for my enemy. The character model I aimed to use as the enemy is from the video game Paragon, created by Epic Games. Heavily inspired from the character Kallari, these actions utilize the provided animations and effects given as part of the character's assets, given out by Epic Games for use in Unreal Engine. Since the actions were tied to the given assets, I researched the Kallari character's abilities through gameplay upload to YouTube and designed and implemented the actions from it, as well as created one from my own design.

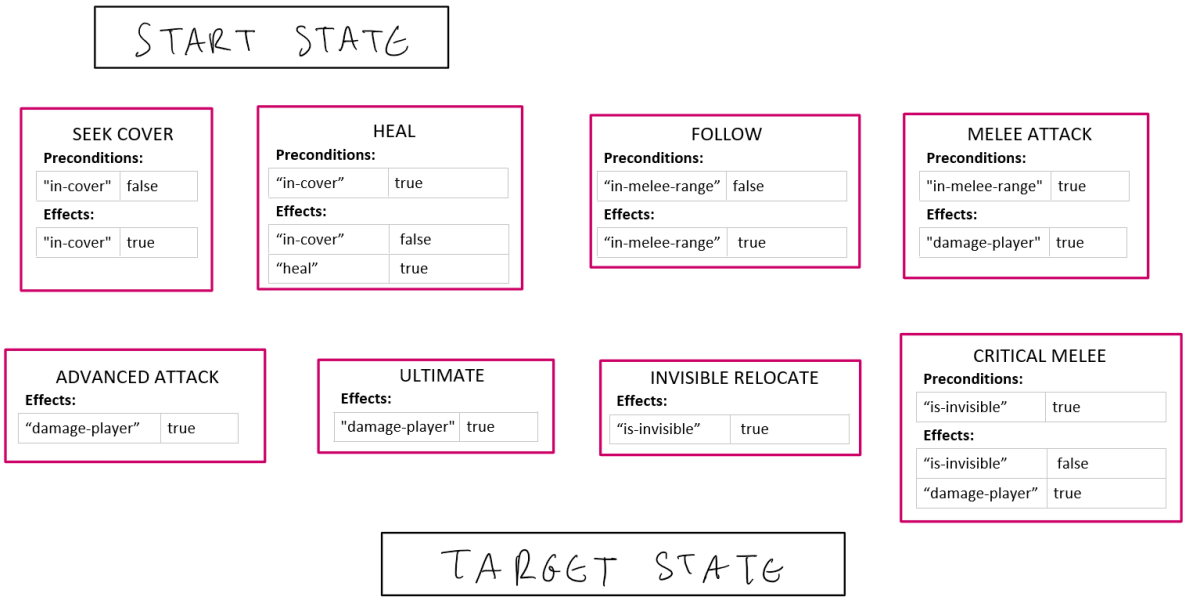


Figure 3.2: *Goal-oriented action planning actions with their preconditions and effects*

From figure 3.2, I designed eight actions which utilize the world state's flags to enable and disable the relevant ones on completion of that action. Actions “Advanced Attack” and “Ultimate” do not have any prerequisites and are allowed to execute depending on other prerequisite conditions such as ability cooldown. However, the following sequences rely on their previous actions before being allowed to execute:

- “Seek Cover” action is completed before “Heal”
- “Invisible Relocate” action is completed before “Critical Melee”
- “Follow” action is completed before “Melee Attack”

These three action sequences are the main sequences that the GOAP system can determine and execute, as well as using the single action sequences “Advanced Attack” and “Ultimate”.

3.3 Machine Learning

Since machine learning has many practical methods of being used and implemented, I decided to also try to use it within my artefact. The main goal of the machine learning (ML) would be to provide a new final cost value for the goal-oriented action planning (GOAP) actions. To be able to do this, the machine learning would require a model that contains previous data on what the appropriate outcomes should be, also known as training data. Once the machine learning has training data and has been trained on the given training data model, it will be able to provide you with a new appropriate final value (an output), based off of given values (inputs).

In my system, I designed the machine learning with a range of inputs that are appropriate and relevant to a single GOAP action and a single output that will create a new GOAP cost value. The inputs are based on attributes of a single GOAP action since at runtime, an action will track and store its inputs which can then be used and extracted at any time.

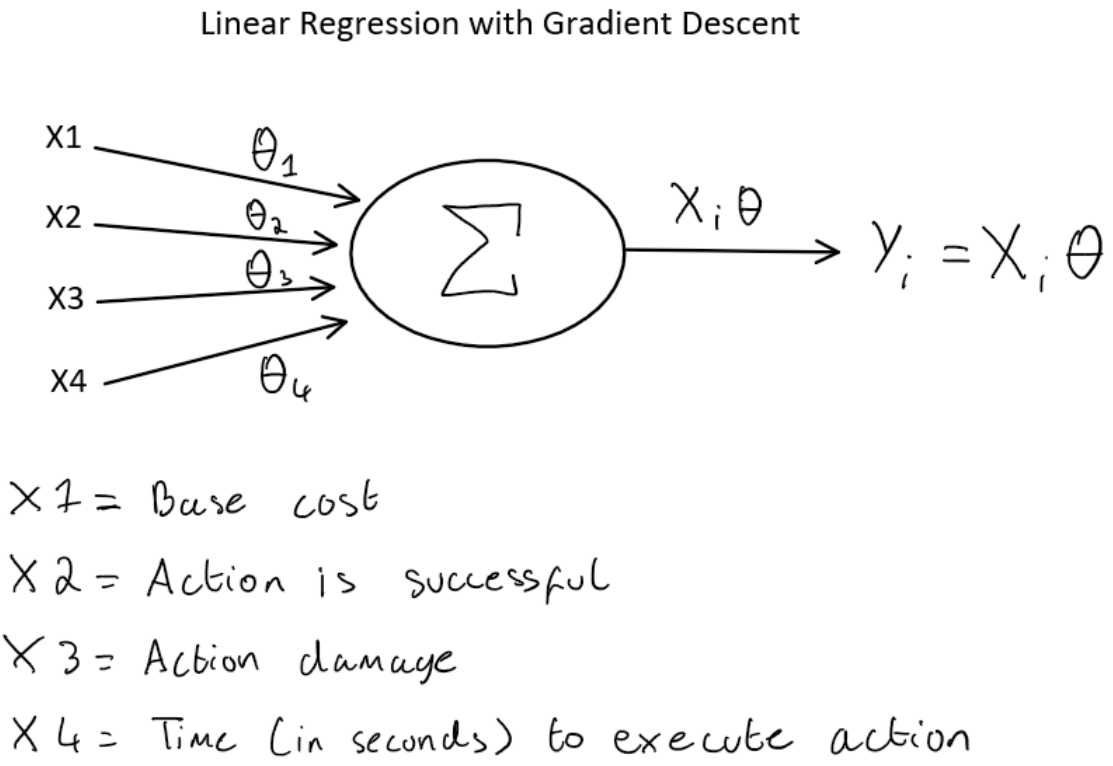


Figure 3.3: *Machine learning model outline*

Shown in figure 3.3 is an overview of the machine learning diagram I designed with the inputs and the output. The model takes in 4 inputs which are an action's base cost (the heuristic value assigned by me), if the actions was a success or not, how much damage the action dealt during the last execution and the time in seconds to complete the action during last execution. The model also outputs a single output value which is the final cost of the GOAP action.

I also chose to use the linear regression with gradient descent method for the design and implementation as this method can be computationally efficient compared to other methods, especially when my training data model can contain a massive amount of data, such as up to 1000 data entries.

3.3.1 Random Range Machine Learning Model

A "random range" machine learning model was used to generate goal-oriented action planning (GOAP) action data within given specific thresholds. The aim with this model would be to create the data, import the data into the artefact, play a session with the newly trained data and then tweak the previous thresholds to generate the training data if necessary.

To tackle this task and create the data, the model was implemented inside python as a separate element alongside the artefact. The pseudocode in figure 3.4 gives an outline of how the system works to initially create a list of actions that have random values within the threshold ranges for each property, and then saves the list of actions to a comma-separated values (CSV) file. The CSV file then could be easily placed alongside the built artefact for distribution and use.

```
determine_final_cost(basecost, successful, damage, executionTime)
{
	rnd_final_cost = create initial random value for final cost between range
	if successful
		multiply rnd_final_cost by 0.x value
	else
		multiply rnd_final_cost by 1.x value
	// A good estimation time to run the action such as less than 6 seconds
	if executionTime is less than a good estimated execution time for actions
		multiply rnd_final_cost by 0.x value
	else
		multiply rnd_final_cost by 1.x value
	// Only reward the final cost with small bonus if it has damage
	if damage is more than 0
		multiply rnd_final_cost by a 0.x value
	// Return the final cost
	return rnd_final_cost
}

// The amount of actions to generate and store in .csv file
action_count = 1000
all_actions = list()
for i in action_count
	create an action class to hold data
	create random value between range for base cost
	create random value between range for if actions was successful
	if successful
		create random value between range for damage
	else
		damage equals zero
	create random value between range for action execution time
	determine_final_cost(basecost, successful, damage, executionTime)
	add action to all_actions list

save all_actions list to .csv file
```

Figure 3.4: Pseudocode for random range CSV file generator

Since the method shown in figure 3.4 allows for quickly generating a massive amount of data in a short time, it sped up my total testing time and I was able to implement, tweak and fine tune this model for a combat style that allows for the GOAP actions to be updated with appropriate costs within two executions, potentially causing them to not be used again if they were executed badly, through missing the action and performing no damage or being a success. Through my testing with this new model, I noticed this and was happy with the results and that the two models have such a clear difference in their style.

3.4 Implementation Tools

I chose to use the Unreal Engine as my game engine to implement the systems since the engine is designed to use C++, which is renowned for being fast and allows for running more computational complexity at a time, compared to Unity. The engine is commonly used in professional video game development for its rigid implementation structure and its long history of previously successfully released video games, as well as its large community of developers and support. Using this engine is a clear choice and will provide the best results for the designed implementation.

I also chose to use Python to implement the machine learning section as the language is written in C++ which comes with the speed benefit, as well as already having many machine learning packages, such as SkLearn, NumPy and Pandas, which can simplify the process. Like the Unreal Engine, Python also has a large community of developers who use the language which provide a lot of resources online for new developers to learn and use.

4 Implementation of an adaptable virtual character

When I began implementation of the artefact, I decided to use Unreal Engine version 4.25 which had released a couple months before. Using this version, I would be able to use plugins and assets that should be supported as developers and creators had time to make them work with this version.

I began with the base Unreal Engine third-person template which included a base mannequin model with simple code to control a character's movement to navigate around a scene. I quickly imported all of the relevant Paragon character assets and configured the main player character and enemy character, and implementing their animation controllers, as well as their functional code to perform their given abilities.

When creating my main player character, I used the same abilities and animations given as part of the Wukong character pack. The player has the ability to switch between two states: offensive and evasive. The player also has two abilities that will perform different actions depending on the state. If the player is in an offensive stance, the first ability will shoot fire where the player is aiming and dealing a medium amount of damage, while the second ability which can only performed while in the air will slam down to the ground and deal damage in an area around the landing, while also stunning the enemy. However, if the player is in evasive stance, the first ability will launch the player in the air, dodging any instances of damage from the enemy, while the second ability will allow the player to walk on clouds while in the air.

Once I had implemented a basic third-person game with a player character that can perform their abilities, I then moved on to implementing my previously designed systems.

4.1 Goal-Oriented Action Planning (GOAP)

To assist me in implementing goal-oriented action planning (GOAP), I took found an open-source Unreal Engine plugin called "GOAP NPC Plugin for Unreal Engine", created by Narratech (Original source located at github.com/Narratech). The plugin already contains a base framework for creating GOAP actions, and also contains an A* pathfinding planner, on top of being integrated with Unreal Engine, using C++ instead of using blueprints. I used this plugin as a base and applied my own modifications and extended the system to fit the purpose of the project.

As a requirement for the plugin when implementing a character that is controlled through a GOAP system, I created a class (`AGOAPAIController`) that inherited from the plugin's `AGOAPController` class. This `AGOAPController` class inherits from the Unreal Engine's `AAIController` class which allows for a `ACharacter` character to be controlled by this controller very easily by setting the `AIControllerClass` variable to my class `AGOAPAIController`.

Implementation and Creating Actions

I created a base class called `UPBGOAPAction` that all of my custom GOAP actions would inherit from, which inherited from the plugin's `UGOAPAction` so that my code will integrate with the plugins. When creating a new action that inherited from `UPBGOAPAction`, there are a handful of variables and two main methods that are used and accessed by the GOAP system.

When implementing a new action, the action needs to set its vital variables inside the constructor that tell the system what this action is, as well as its costs, preconditions, and effects.

```
UAction_Follow::UAction_Follow()
{
	// Set the name of the action
	name = "follow target";
	// Set the BaseCost of the action. A constant value
	BaseCost = 5.0f;
	// Set actual cost to the BaseCost
	cost = 0.0f;
	// Set the target type of this action
	targetsType = AProjectBossCharacter::StaticClass();
	// Create preconditions for action
	preconditions.Add(CreateAtom("in-melee-range", false));
	// Create effects to world state once action is complete
	effects.Add(CreateAtom("in-melee-range", true));
}
```

Figure 4.1: *Example action constructor that sets the information needed for the planner*

Figure 4.1 shows the code for implementing the "Follow" action, such as setting its name to its preconditions and effects. The `UPBGOAPAction::BaseCost` variable is used as a constant value that is used as a baseline and is determined by myself, the designer. When the planner wants to get that action's cost, it would add the `BaseCost` and the `cost` variables together to get the actual cost of that action.

The methods that get used in the GOAP system are `UGOAPAction::checkProceduralPrecondition(APawn* p)` and `UGOAPAction::doAction(APawn* p)`. The purpose of the `checkProceduralPrecondition` method is to perform additional validation to the precondition of the action, such as check that the action is not on a cooldown or check that the distance between the enemy and player is not larger than expected. The `checkProceduralPrecondition` function returns a boolean, which communicates to the GOAP planner if the action has passed all procedural preconditions such as cooldowns. If the planner finds that the action satisfies the precondition in the world state as well as satisfying any procedural preconditions, then the action can be used in the sequence the planner is attempting to determine. The other function, `doAction`, is used to implement functional code to perform that action. For example, in the "Follow" action's `doAction` code, would be pathfinding code to walk towards the target of the action. The function also returns a boolean which is used to specify if the action has completed in its `doAction` aim.

Setting the World State

Once all of my actions were created, the next step was to configure and set the world state of the GOAP system. In my `AGOAPAIController` class, in the `Tick(float deltaTime)` function is where the GOAP system executes. The system is called once every frame which allows it to either execute the planner to determine a new sequence of actions or to execute the current determined plan until it is completed.

```
AGOAPAIController::Tick(deltaTime)
{
	executeGOAP()
	if (planCompletedExecuting)
	{
		DetermineNextWorldState()
	}
}

AGOAPAIController::DetermineNextWorldState()
{
	if (enemy.Health < HealthThreshold && Heal.Cooldown <= 0)
	{
		return TargetState = “heal”
	}
	else
	{
		return TargetState = “damage-player”
	}
}
```

Figure 4.2: *Pseudocode that describes how I implemented how to determine the next world state target*

Once the current plan has finished executing, the target world state will have been met. Once this is detected in the `Tick` function, the controller then determines the next world state. As shows in the original GOAP design, the world state target will always be trying to set the flag “damage-player” to true. However, if the enemy currently meets the “Heal” action procedural preconditions, preconditions that are unrelated to the world state such as a cooldown or a health threshold to perform the action, then the target world state will set the flag “heal” to true and perform the “Seek Cover” and “Heal” actions in that sequence. Figure 4.2 shows pseudocode of how this simple system works.

4.2 Machine Learning in Python & Unreal Engine

For my implementation, I decided to use Python for the implementation as the language already contains many packages with functions for machine learning such as SkLearn and TensorFlow. As I initially approached the implementation of my machine learning design, I realised that I would require a method to bridge between Unreal Engine C++ code to Python. With this problem, I found and utilized a plugin for Unreal Engine called UnrealEnginePython. This plugin is already popular for previous versions of Unreal and helps to run Python code from C++ using reflection, and also takes away any problems of interfacing between the two languages.

With the ability to communicate between C++ and Python, I began to implement the linear regression algorithm and split the code up into two main functions. The first function, called `init_ml()`, would be called once the enemy character is created, handling initialization of the machine learning and performing the code to load the training data and creating a theta value to be used to create any new costs in a goal-oriented action planning (GOAP) action. The second function, called `predict_cost()`, would then be called once a sequence of actions in the GOAP system is complete to update all actions with a new cost before replanning the next sequence of actions. The function takes four arguments which are the four inputs into my machine learning algorithm, with the output value, the final cost, being returned from the function. This is one of the simplest ways of implementing this system since the workload that requires the most processing only done during start-up which allows for the least processing to be done whenever the GOAP actions require updating to a new cost.

Creating a Model from Gameplay

After implementing my machine learning code, the next step was to implement a system to extract and store training data from my play sessions with the enemy. The enemy was already able to use the GOAP system to perform the same sequence of actions at runtime and call the python code, however no training data was being provided to the ML system which causes it to not function correctly. To fix this, I created to system to track and store when the GOAP system had completed a sequence of actions. All sequence of actions would get stored until the end of the session at which point either the player has died, or the enemy had. Once that point had been reached, the system would then iterate through each sequence and each action in that sequence to extract the required values from the ML algorithm. The system would also extract the current total cost of the action and store that along with the other data. Once all data was stored, it would then get extracted and saved into a comma-separated values (CSV) file ready to be loaded by the ML system.

However, during the training sessions, the final cost needed to be adjusted and influenced to be an appropriate value, since this data will be used in my machine learning system. To do this, I stored the health of both the player and the enemy once the action had started executing. Once the action had completing execution, I would then get the health values of both characters again and determine the difference between the two. Once determined, the cost would be updated by multiplying the differences, after dividing by 10. The pseudocode in figure 4.3 also represents my method to update the code to train this model.

```
// Get the health difference of the enemy
enemyHealthDifference = enemyEndHealth – enemyStartHealth
// Get the health difference of the player
playerHealthDifference = playerEndHealth – playerStartHealth
// Determine cost multiplier by multiplying differences divided by 10
// to give a multiplier between 0 and 5
costMultipler = (playerHealthDifference / 10) * (enemyHealthDifference / 10)
// Don’t allow a less than 0 value
if (costMultiplier < 0) {
       costMultiplier = 0
}
// Update final cost with the multiplier
cost = previousCost * costMultiplier
```

Figure 4.3: *Pseudocode for determining training final cost*

With this method in figure 4.3, I was able to create a final cost value that would be multiplied by a 1.x value if the boss had lost more health than the player causing the cost to become bigger and also multiply the cost by a 0.x value to reduce the cost. When the cost is reduced, the GOAP planner will favour that action in comparison to the other actions that have a higher cost. This initial model will favour actions that are able to deal a big amount of damage to the player and will punish the actions that are not able to do that, either because the player was able to dodge the damage, or the enemy missed while executing that action.

With the ability to now save training data to a file once a play session has finished, I then played the artefact a handful of times to produce the data. I made sure during each session to fulfil the preconditional requirements of each action, such as being a certain distance away, so that all actions were executed multiple times, as well as ensuring I was able to dodge the damage of some of the actions including to also receive damage. By doing this, the produced data would include all scenarios possible, as well as creating final costs for all of the possibilities too.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	baseCost	didSucceed	damage	averageExecuteSeconds	finalCost												
2	10	10	10	50	5	50	50	5	20	20	50	5	50	50	5	50	50
3	1	1	1	1	0	1	1	0	0	1	1	0	1	1	0	1	1
4	60	0	60	40	0	40	40	0	250	150	40	0	40	40	0	40	40
5	0	6.01729	0	0.592739	3.573378	0.592739	0.592739	3.573378	2.812618	1.164672	0.592739	3.573378	0.592739	0.592739	3.573378	0.592739	0.592739
6	10	10	10	79.066978	5	79.06698	79.06698	5	19.06698	19.06698	79.06698	5	79.06698	79.06698	5	79.06698	79.06698

Figure 4.4: *The resulting training data comma-separated values (CSV) model created from gameplay sessions*

Figure 4.4 shows a small handful of values from the resulting comma-separated values file that is produced from the artefact after implementation and play sessions.

4.3 Logging inside Unreal Engine

A useful component of any game engine is the ability to output any messages at runtime to a console and a logging file. Inside of the Unreal Engine, the engine provides the ability to log out messages under categories, custom made or using the included categories with the engine. I utilized this feature and implemented my own categories and included log messages related to the goal-oriented action planning system and the machine learning system, logging vital information about these systems like the current GOAP sequence and the new costs of GOAP actions set by the machine learning model.

With this, I am able to analyse and read a game's session purely from the logging file that is created and outputted by the engine. As I aim to hand out my artefact to participants, I will optionally ask them to upload their log file which will allow me to compare their answers with the logs and make assumptions about how the participant experienced the artefact.

5. Experimentation & Testing

Once I had completed implementing my artefact, the next stage was to test and experiment with the resulting application to tune and adjust the results towards the main goal of the artefact, adapting to a human player. The main purpose of the experiment is to test if the created agent is able to adapt itself to the current player. As the project is centred around this,

5.1 Testing with Human Players

In my academic question, I specify how my enemy is able to "adapt to a human player". To truly answer this section, I needed to test my artefact on extra human players, excluding myself. The group I wanted to test my artefact on were computer science students from the University of Wolverhampton. The reasons why this would be the best group to test my artefact on are because computer science students are people who most likely to play video games, giving them prior experience playing different types of video games and a level of competency to pick up and play a new video game. The other main reason is that the students would be interested in being able to determine the enemy's play style to defeat the boss through multiple play sessions.

Once a participant had played the artefact for a minimum of five sessions, they were asked to complete a questionnaire form which contains a couple questions about themselves and a handful of questions about their experience of the artefact. With a minimum session requirement, the participant will have had chance to experience a change in the enemy's behaviour, as well as learn different techniques to avoid its attacks. The participant is allowed to play for longer if they wish, I avoided a restriction on the number of games played to allow them to train and use the machine learning model to their playstyle.

The questionnaire contains a couple questions related to the participant as a way to analyse and understand how experienced they are with video games. If a participant believed the enemy was able to adapt but has not played a video game before, then their answers, when compared alongside other participants, would help to determine what skill level a player has for the goal-oriented action planning and machine learning systems to be able to adapt and beat that participant.

An important question the questionnaire asks near the start is if the participant had checked the "Enable Participation" tick-box on the artefact's main menu. This tick-box will track the user's play sessions and update the generated ML model with the participant's play data, making a customized model to target that participant. This mode is optional to the participant and explains to the participant that their data will be tracked and used to update a machine learning model.

The other section of the questionnaire contained questions related to the participant's experience with the enemy. The questions in this section focused on the core features of the enemy and centred around if they noticed a difference of the enemy's behaviour. Primarily, the focus of this section is on a set of statements that the participant could choose a number between one and five. One meaning they strongly agree up to 5 meaning they strongly agree, with three meaning the participant does not agree or disagree. Figure 5.1 shows some of the agree and disagree statements included in the questionnaire.

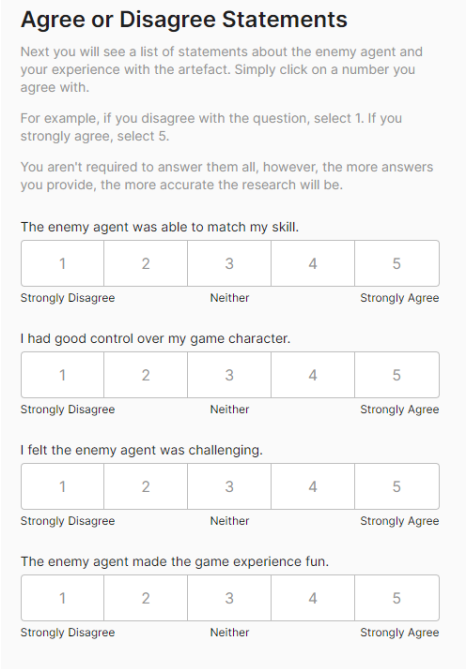
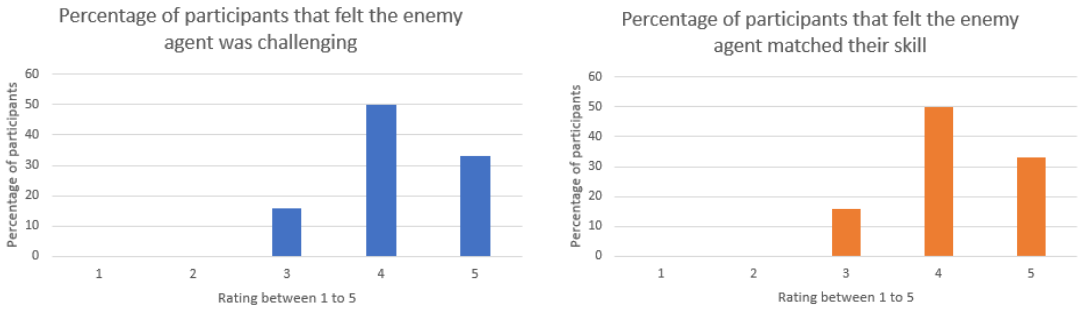


Figure 5.1: *Agree and disagree statements within the participant questionnaire*

Finally, the last question in the questionnaire asks for the user to upload their log file which contains debug information as well as statistical information from the game's GOAP system and machine learning system. This request is completely optional since some participants would not want their logging data to be tracked or seen which I abide by.

5.2 Questionnaire Results

In total, I was able to get 6 participants to play the artefact and complete the questionnaire. From the participant's results, I was able to determine that with the machine learning section, the enemy was able to prove itself and learn to play against a certain participant. In figures 3.9, according to 50% of the participants, they felt that the enemy was able to match the participant's skill and prove to be a challenging opponent.



Figures 5.2 (left) & 5.3 (right): *Percentage of participants that matched skill and though the agent was challenging*

Also, from the questionnaire results, I was able to have participants that actively to semi-actively play video games, from every day to one or twice a week. With these participants, I feel the questions and artefact benefitted from being answered by players who have a history of playing other video games that contain artificial intelligence, giving the player's a certain expectation of the artefact's agent. A majority of 66% of the players felt that the artefact's enemy agent benefitted from being slightly larger than the players, since they felt intimidated from the size.

Finally, a majority 66% of participants were able to defeat the enemy, with the machine learning section of the artefact enabled. This means that only 33% of players were not able to defeat the enemy, however this did not impact negatively on the final feelings. This is because all participants rated their experience of the artefact a 5 or above, as seen in figure 5.4.

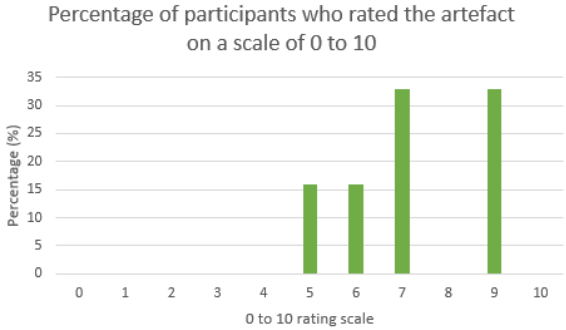


Figure 5.4: *Percentage of participants who rated the artefact on a scale of 0 to 10.*

6. Conclusions

During the course of the project, I was able to discover and implement a goal-oriented action planning (GOAP) system that utilizes machine learning to update its action costs to adapt itself and provide a challenging but fun experience for the player. From the resulting artefact, participants were able to train and allow the enemy to adapt itself to the player, which the participants found to be an exciting experience. As the aim of this project was to investigate and implement an enemy that is able to adapt, I feel like I was able to reach to main aim of my project and produce a resulting enemy with a range of actions that will execute depending on previous experience. With the inclusion of machine learning, it helped to assist the adaptability of the enemy is a big way as otherwise, the enemy would not change its actions.

From the participant's questionnaire results, their overall sentiment was that the enemy was able to match their skill by adapting itself. I did not receive any answers that were a surprise to me such as participants not receiving the expected outcome or being unhappy with their session with the artefact. However, I was surprised that a couple participants were unable to defeat the enemy agent. Because of this, the enemy agent could still benefit from more refinement and tuning to not be too difficult to beginner players. This could be achieved through tweaking with the machine learning training and tracking data to allow for more room for errors in the enemy's actions.

The main aim of the project is to focus on creating an enemy agent inside of a video game environment that uses a different type of artificial intelligence. From the results, 50% of the participants rated the enemy as a 4, on a scale from 1 to 5, for the agents difficultly as well as its ability to match to the participant's skill. These questions were aimed at the artificial intelligence system's ability to adapt itself and how it performed with the adjustments from the machine learning system. These results reach the aim with the project's question of being able to have an adaptable AI system.

As a result of the whole project, I am confident that I was able to match my project's aim, while producing a quality artefact with visible outcomes when opting to participant in the machine learning tracking. The project set out to determine if there were available systems that can be used to improve and/or train an enemy to match a player's skill and I feel my artefact was able to achieve that. I expected that the enemy would be able to match the player's skill and managed to achieve this outcome with most of the participants.

7. Evaluation of Product

7.1 Product Evaluation

The final product of the project was the complete artefact with an enemy that is able to adapt to the player, learning from its current actions and changing its future actions on previous sequences that was not successful. As the enemy was able to be trained by the participants, when they enabled the machine learning system, the resulting performance managed to match the skill of a more experienced player but remained at the same difficulty for participants who had less experience with video games. Most participants were able to defeat the enemy with their trained data and enjoyed their experience with the artefact. This shows that the artefact was able to achieve its goal of adapting to the player and providing a fun experience.

7.2 Sources Evaluation

The sources I utilized in my literature review were found mostly from using Google Scholar, IEEE Digital Library and O'Reilly Books. The sources I required were very easy to find on IEEE and O'Reilly since they specialize in displaying computer science papers and articles, and already contained a lot of papers around video games and artificial intelligence. I especially liked reading and using the Game AI Pro series of books by Steve Rabin as the main focus of my artefact was to learn and experience new ways of implement artificial intelligence and these books game be a big insight on new and different methods. I also found a lot more sources related to video game implementation where I picked up a couple new bits of information but was not able to use a quote that related to a point I wanted to talk about in my review. Overall, I am happy with my use of sources but wish I could have found more informative papers, articles, or books around advancements in artificial intelligence or about a system being able to adapt to a human player.

7.3 Self-Reflection

Looking back over my project, I can clearly see areas in which I feel I did not do well enough or, if I were to redo the project, could do with much more refinement and focus. However, I do feel that there are areas of my project that contain elements that I feel did well and make me proud of the resulting code and overall result.

7.3.1 Overall Artefact and Results Reflection

I am very pleased with the resulting artefact I produced, even though it contains some areas that, with more time, I would want to improve or rewrite, leading to better performance. With every software release, there are bugs and problems but with more time and work, developers can fix these, which is the same as how I feel with my project.

The biggest issue that I wish I could fix with my artefact would be the massive lag/freeze once the enemy is created. The reason this is happening is because 3 different systems (The GOAP, machine learning and UnrealEnginePython systems) are attempting to be started at the same time, causing a lot of work for the system running the artefact. Once method to mitigate this would be to move the UnrealEnginePython start-up to run and initialize as the level is loaded, since 75% of the freeze is from this plugin.

Also, another issue with the artefact is the prerequired installation of Python which is used with the UnrealEnginePython plugin. Because of this, if a participant has not installed it, they were required to download and install it first before being able to launch the game at all. While this is

minor, it still affects the overall user experience of the project and slowed down and impacted my overall participants, since some were not willing to download and install extra software.

7.3.2 GOAP Reflection

Overall, I really enjoyed implementing the GOAP system, with the assistance of the GOAP NPC plugin, is a really unique way to implement non-playable characters. If I were to implement it all by myself, I feel I would have struggled in certain areas such as implementing the A* planner since the overall system was new to me. I am very happy with my additional `UPBGOAPAction` class too, which includes additional functionality to a GOAP action to easily get data on the action related to the machine learning system.

Once area in the GOAP system I feel would be a good improvement would be to include more actions of the enemy. Since I purposely restricted myself on the number of actions, the enemy can only utilize those, leading to quite repetitive gameplay after a handful of sessions. With the addition of more actions, simple or advanced ones, the gameplay could remain fresh for longer leading to the player enjoying and experiencing the enemy for longer.

7.3.3 Machine Learning Reflection

The resulting machine learning system performs well, with some exceptions. The first trained model, using GOAP data from my own play sessions, did not perform as well as I had expected since I did not have enough training data. However, when the machine learning system utilized the “RandRange” model, the results did improve, and the model was able to provide fun and engaging combat between the enemy and player.

8. Product Management

To properly project manage my project, I utilized many tools such as using project Kanban boards to property track and manage any tasks required during the implementation and testing stages, as seen in figure 8.3 and figure 8.4. However, before that stage, I made sure to property plan out all of my stages of the project, from defining the project, preparation, implementation, and final evaluation, I made sure to plan out the estimated time frames of each stage in figure 8.1 which helped me to make sure I stayed on track and on time to complete the project on time.

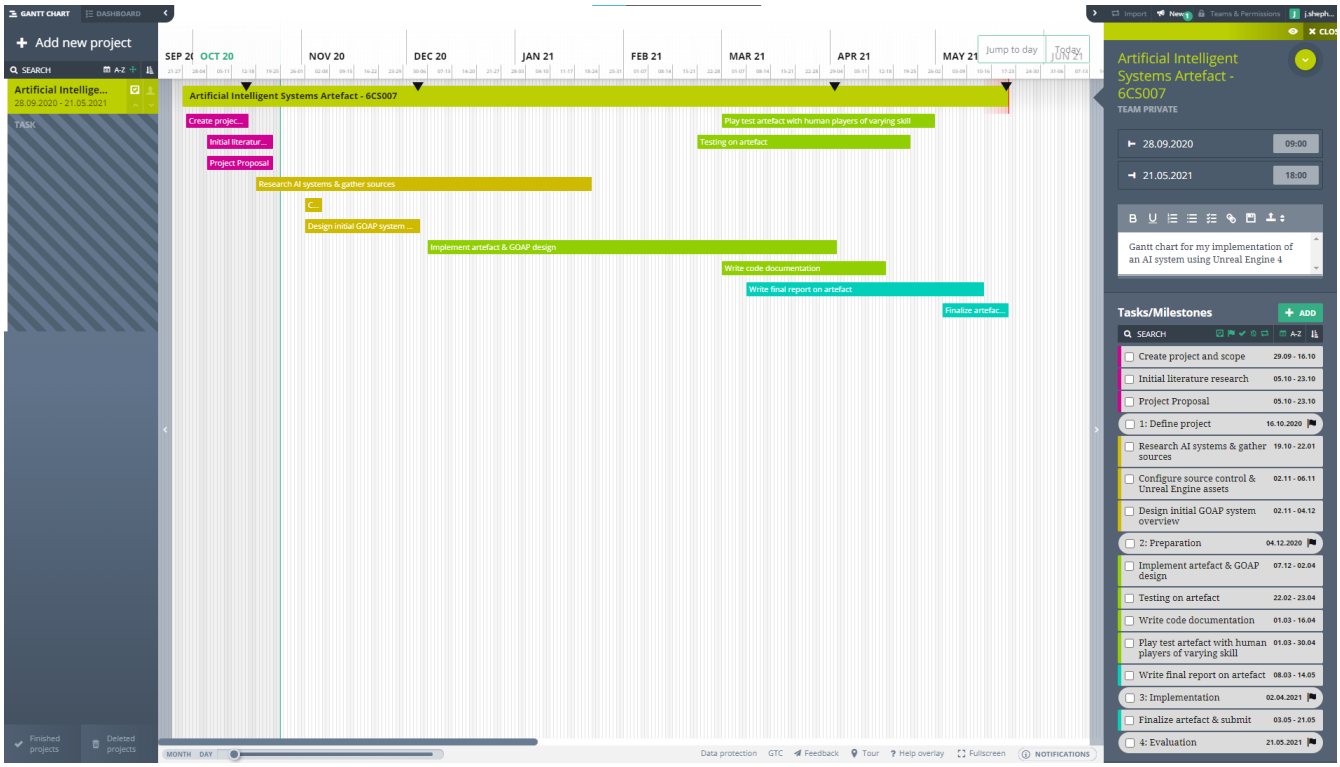


Figure 8.1: *Project gantt chart with time frames for each section*

During the project proposal stage, I made sure to make as many notes as possible on potential topics and relevant techniques I would be able to use for my project. I initially researched videos on the goal-oriented action planning (GOAP) system and made sure to make notes detailing vital information, as seen in figure 8.2

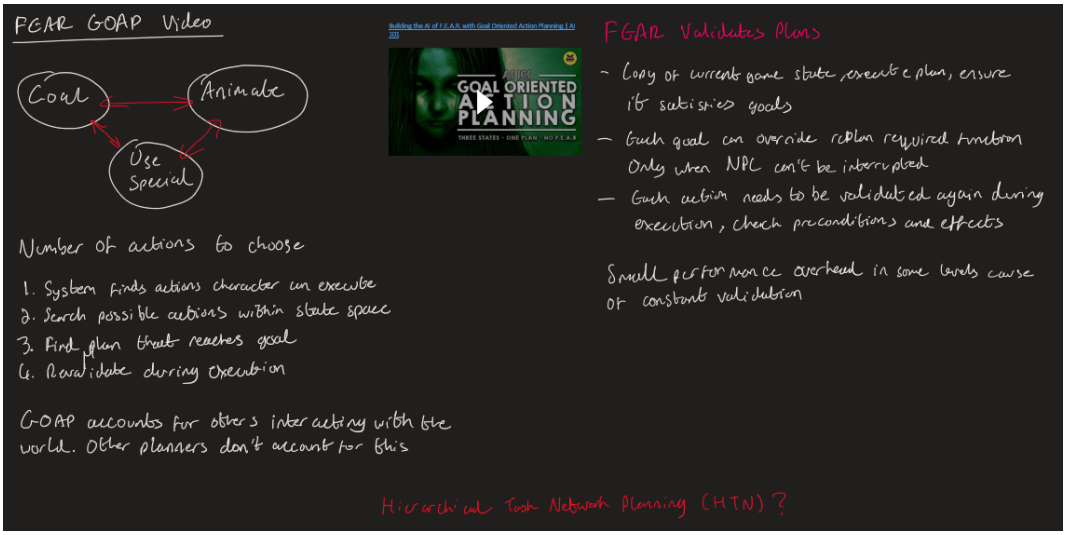


Figure 8.2: *Goal-oriented action planning (GOAP) research notes*

During implementation, I made sure to use a Kanban project board, which allows you to break up the whole process into mini tasks, which are then able to be dragged and moved into different columns such as “To do”, “In progress”, and “Complete”, when they are in their respective stages. Figure 8.3 shows how I broke up machine learning tasks, GOAP tasks and general artefact tasks and organized myself by working on the tasks and moving them around on the board.

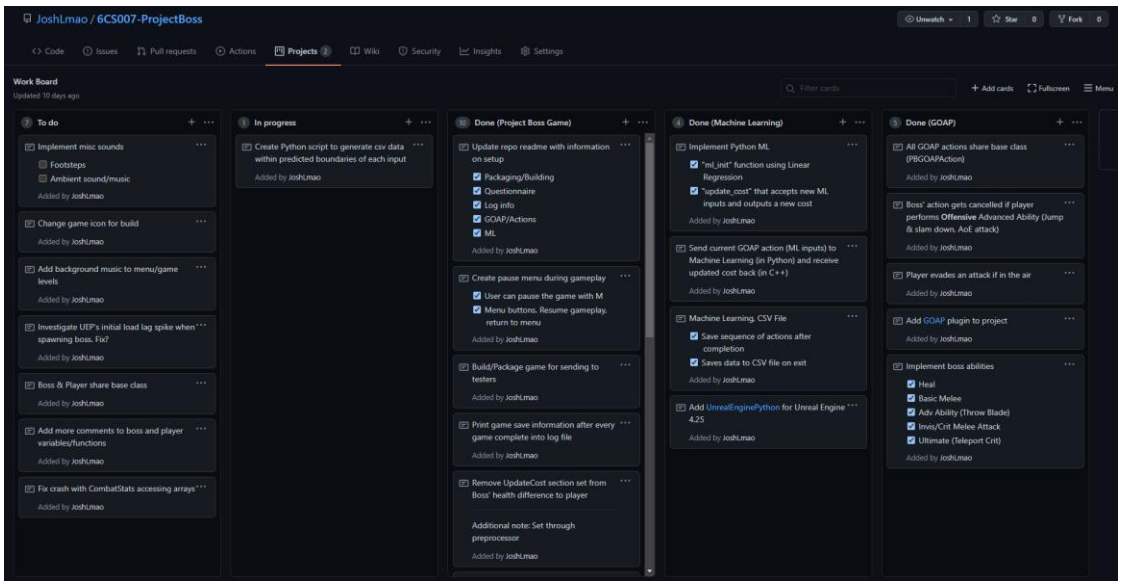


Figure 8.3: *Work Kanban board with all implementation tasks, split up by GOAP and ML systems.*

I also did the same style Kanban board to plan and track my progress during testing, as seen in figure 8.4. I, again, broke the testing tasks up which allowed me to easily visualize what I had completed and what tasks were remaining.

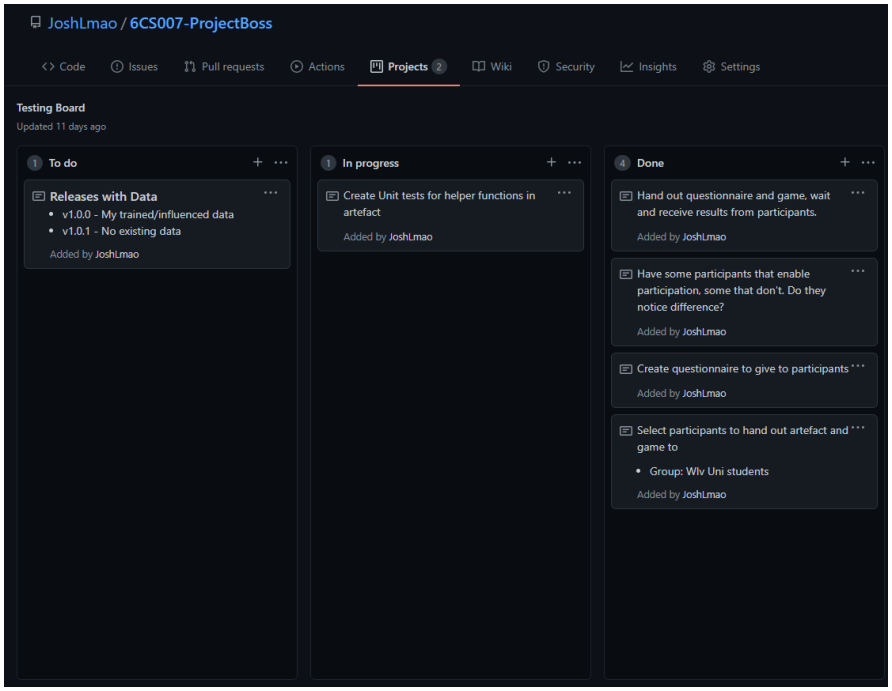


Figure 8.4: *Testing plan Kanban board with tasks*

9. References

Bakkes, S., Spronck, P., & van den Herik, J. (2009). Rapid and Reliable Adaptation of Video Game AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 93-104.

Benedikte Mikkelsen, C. H. (2017). *Ethical Considerations for Player Modeling.*

Bertens, P., Guitart, A., Chen, P. P., & Perianez, A. (2018). A Machine-Learning Item Recommendation System for Video Games. *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1-4.

Bimbraw, K. (2015). *Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology,.*

Bourg, D. M., & Seemann, G. (2004). *AI for Game Developers.* O'Reilly Media Inc.

Brown, J. A., Cuzzocrea, A., Kresta, M., Kristjanson, K. D., Leung, C. K., & Tebinka, T. W. (2017). A Machine Learning Tool for Supporting Advanced Knowledge Discovery from Chess Game Data. *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 649-654.

Champandard, A. J. (2017, June 26). *Artificial Intelligence Half-Life SDK: Retrospective*. Retrieved from sudonull: https://sudonull.com/post/69458-Artificial-Intelligence-Half-Life-SDK-Retrospective

D. Johnson, J. W. (2001). *10th IEEE International Conference on Fuzzy Systems.* (2 ed.). Melbourne, Victoria, Australia.

DaGraça, M. (2017). *Practical Game AI Programming.* Packt Publishing.

Ford, M. (2018). *Architects of Intelligence.* Packt Publishing.

Georgievski, I., & Aiello, M. (2015). HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 124-156.

Gill, S. (2004). *Visual Finite State Machine AI Systems*. Retrieved 10 17, 2020, from https://www.gamasutra.com/view/feature/130578/visual_finite_state_machine_ai_.php#:~:text=Finite%20State%20Machines%20(FSMs)%20are,handling%2C%20UI%2C%20and%20progression.

Hawkins, A. J. (2020, Octover 19). *Americans still don't trust self-driving cars.* Retrieved from The Verge: https://www.theverge.com/2020/5/19/21262576/self-driving-cars-poll-av-perception-trust-skepticism-pave

Ian Millington, J. F. (2009). *Artificial Intelligence for Games* (2nd ed.). CRC Press.

Isaac Skog, P. H. (2009). IEEE Trans. Intel. Transp. Syst., vol. 10, no. 1. *IEEE*, 4-21.

J. Contreras-Castillo, S. Z.-I. (2019). Autonomous Cars: Challenges and Opportunities. *IT Professional*, 6-13.

K. Jo, J. K. (2015). Development of Autonomous Car—Part II: A Case Study on the Implementation of an Autonomous Driving System Based on Distributed Architecture. *IEEE Transactions on Industrial Electronics*, 5119-5132.

Lanier, L. (2019). *Video Games Could Be a $300 Billion Industry by 2025 (Report)*. Retrieved October 24, 2020, from https://variety.com/2019/gaming/news/video-games-300-billion-industry-2025-report-1203202672/

M. König, L. N. (2017). Users' resistance towards radical innovations: The case of the self-driving car. *Transportation Research Part F: Traffic Psychology and Behaviour*, 42-52.

Mac Namee, B. (2004). *Proactive Persistant Agents: Using situational intelligence to create support characters in character-centric computer games*.

Mac Namee, B. (2004). *Proactive Persistent Agents.* Dublin.

Mike Loukides, B. L. (2018). *What Is AI?* O'Reilly Media, Inc.

Mitchell, J. (n.d.). *Goal-Oriented Action Planning Research*. Retrieved from Jared Mitchell: https://jaredemitchell.com/goal-oriented-action-planning-research/

Nareyek, A. (2004). AI in Computer Games: Smarter games are making for a better user experience. What does the future hold? *Queue, 1*(10), 58-65.

Neufeld, X., Mostaghim, S., Sancho-Pradel, D. L., & Brand, S. (2019). Building a Planner: A Survey of Planning Systems Used in Commercial Video Games. *IEEE Transactions on Games*, 91-108.

Olsen, K. A. (2020). The Case Against Autonomous Cars. *IEEE Potentials*, 25-28.

Ontañón, S., & Buro, M. (2015). Adversarial hierarchical-task network planning for complex real-time games. *Proceedings of the 24th International Conference on Artificial Intelligence*, 1652–1658.

Orkin, J. (2006). Three States and a Plan: The A.I. of F.E.A.R.

Orkin, J. (2008). Applying Goal-Oriented Action Planning to Games.

R Hussain, S. Z. (2019). Autonomous Cars: Research Results, Issues, and Future Challenges. *IEEE Communications Surveys & Tutorials, vol 21, no. 2*, 1275-1313.

Rabin, S. (2013). *Game AI Pro.* A K Peters/CRC Press.

Rabin, S. (2015). *Game AI Pro 2.* A K Peters, CRC Press.

Risto Miikkulainen, B. D. (2006). Computational Intelligence in Games. *Computational Intelligence: Principles and Practice*, 155-191.

Rob Thomas, P. Z. (2020). *The AI Ladder.* O'Reilly Media, Inc.

Russell, S. (2016). *Artificial intelligence: a modern approach.* Pearson Education.

Sebastian Welsh, Y. P. (2005). Information-Oriented Design and Game AI.

Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., & Postma, E. (2005). *Adaptive game AI with dynamic scripting.* Springer Science & Business Media.

Togelius, J. (2019, April 17). *How Games Will Play You*. Retrieved from The Ethical Machine: https://ai.shorensteincenter.org/ideas/2019/4/7/how-games-will-play-you

Y. Li, X. X. (2020). Adaptive Square Attack: Fooling Autonomous Cars with Adversarial Traffic Signs. *IEEE Internet of Things Journal*, 1-1.

Zhang, P.-F., Su, Y.-J., & Wang, C. (2007). Statistical Machine Learning Used in Integrated Anti-Spam System. *2007 International Conference on Machine Learning and Cybernetics*, 4055-4058.