Viability of Machine Learning for enemies in Video Games

An analysis of enemy non-player character behaviors created with reinforcement learning

Mattias Larsson
William Örnquist

Department of Computer and Systems Sciences

Degree project 15 HE credits
Computer and Systems Sciences
Degree project at the bachelor level
Spring term 2022
Supervisor: Mirjam Palosaari Eladhari
Swedish title: Rimligheten av att använda maskininlärning för fiender i datorspel

Stockholm University

Abstract

Video game enemies require behaviors, currently there are various well researched, documented and accepted methods to create these behaviors such as finite state machine and behavior tree method. . An alternative potential method of creating these behaviors is with the use of machine learning. The current advancement and achievements in developing machine learning artificial intelligence are mostly used for other means rather than games. This technique allows video game characters which are controlled by the computer known as ‘non-player characters’ (NPCs) to learn from the environment and act upon it to create both complex and interesting behavior. Because of these achievements, it seems to be possible to create enemy behaviors with machine learning.

Recently, large scale projects for machine learning of artificial intelligence have proven to be very capable of rivaling professional players in competitive video games as players themselves. But is machine learning able to integrate into the game's environments and serve as believable and entertaining enemies against more casual players?

Currently, commercial games that integrate machine learning into interactive characters are exceedingly rare despite the reliability and widespread use of such technology in computer science including graphics in video games. The problem is that there is a lack of research into the reasoning behind the absence of machine learning NPCs in even the most popular games on the market.

This study experiments and researches on the current state of machine learning in terms of creating video game behavior for NPCs to understand its flaws and benefits. The experiment involves creating machine learning enemies in a simple 3D video game with stealth gameplay mechanics. It was created using the ‘Unity’ game engine along with additional components for the implementation of machine learning behavior and more.

The experiment collects statistical data on the performance of the machine learning enemy NPC and a traditionally designed enemy NPC to make a side-by-side comparison to learn of their efficiency. During the process, observational data is also collected to analyze how each NPC behaves in order to determine their level of quality in terms of ‘believability’.

The results of the experiment shows that it is possible to create video game enemies with machine learning. However, the complexity and time consuming effort of using machine learning makes it a difficult process. The machine learning agents are not fully believable from an implied players perspective in comparison to creating behaviors in the standard traditional way. Which is easier, both in terms of complexity and achieving believability.

Keywords

- Video games
- Artificial intelligence
- Machine learning
- Game design
- Believability

Synopsis

Background	The thesis is about machine learning being used to create behaviors for non-player characters in video games, specifically characters that take the role as ‘enemy’. Research area : Digital Games and Simulation
Problem	Commercial video games rarely make use of machine learning to create more complex/smarter artificial intelligence for non-player characters despite the method being widely used and relied on in many other fields within computer science.
Research question	How viable is machine learning for creating believable enemies in video games?
Method	This study makes use of the ‘Unity 3D’ engine to create a game environment and simulate (Simulation method) two types of non-player characters that make use of machine learning and traditional artificial intelligence respectively. The characters are then observed (Observation method) during testing to assess their behaviors.
Result	It is possible to create machine learning enemies, however it produces strange behaviors that are not easy to fix and therefore the process might not be considered viable depending on the game design.
Discussion	This study is limited because the experiment is done in a simple game genre (3D game with stealth elements) and therefore the results may not represent how machine learning agents would interact in other game genres such as real time strategy games. Game developers can use the study to understand if machine learning is useful for their game. Other researchers may use this study to get an understanding of using machine learning for video game enemies.

Table of Content

Acknowledgments

We would like to express our gratitude to supervisor Mirjam Palosaari Eladhari for her valuable advice and guidance during this process.

List of Figures

List of Tables

List of Abbreviations

ML Machine Learning

AI Artificial Intelligence

NPC Non-player character

FSM Finite State Machine

BT Behavior Tree

RL Reinforcement Learning

PPO Proximal policy optimization

ACER Actor Critic with Experience Replay

1 Introduction

1.1 Background

Video game enemies need behaviors to make them behave as intended by the designer. Developers create conditions and actions through scripting to make enemies believable, which can be considered as the traditional method when creating artificial intelligence (AI) in games. A different method of creating AI is with the use of machine learning (ML). A machine learning algorithm creates a model based on sample data it collects from the software environment in order to make predictions or decisions without being explicitly programmed to do so (Riitahuhta et al., 2012). This technique is used in a variety of applications across different areas within computer science such as speech recognition, image recognition, email filtering, and medical diagnosis, which are considered unfeasible or difficult to develop conventional algorithms for to perform said tasks (Hu et al., 2020).

Usually, programming an enemy AI requires the developers to consider the types of events that are reasonable to occur within the game's rules and then programming what the AI should do in specific instances. The behaviors that are needed vary greatly depending on the type of game, it is therefore difficult to generalize the requirements of an AI enemy (Yannakakis & Togelius, 2018).

Machine learning at its current state in video games has gained a foothold in creating advanced AI for controlling playable characters in a ‘player versus player’ environment as a third-party tool. Examples of recent projects of this type of research are ‘AlphaStar’ and ‘OpenAI Five’, which were in development for several years. *AlphaStar* was designed around a competitive real-time strategy (RTS) game called ‘StarCraft’, which has on several events rivaled against and managed to defeat professional players (Arulkumaran et al., 2019). *OpenAI Five* was designed around a 5-against-5 multiplayer online battle arena (MOBA) game called ‘Dota 2’ (Berner et al., 2019). It made an appearance in recent events during e-sport tournaments and has managed to defeat the winners of some of these competitions.

The research area of this study relates to digital games and simulation. Artificial intelligence for video game agents has been extensively researched, one of the core concepts in this area is believability which attempts to measure how fitting the non-player characters (i.e computer controlled characters) is in the game world (Warpefelt, 2016, p. 48).

“In order for the player to achieve a feeling of immersion, the game must draw the player in and cause them to become engaged in the game.” (ibid)

If the game is unsuccessful in achieving believability for its NPCs the game might not be appreciated by the players who are looking for immersion. (Yannakakis & Togelius, 2018, p. 13)

“The key criterion that distinguishes successful AI in commercial-standard games had always been the level of integration and interweaving of AI in the design of the game” (ibid)

Despite the existence of open-source tools like ‘Unity ML-Agents’ (Juliani et al., 2020) which is used for applying machine learning to in-game entities, and the recent achievements of self-learning game AI (Fürnkranz, 2007). This method of developing AI characters in particular sees no widespread use for commercial video games in recent years commonly due to the difficulty of executing and maintaining it (Skarupke, 2020) (Justesen et al., 2019, p. 16).

1.2 Problem motivation

Although machine learning sees widespread use in many subjects such as search recommandation, speech recognition, data mining etc. creating intelligent game agents through machine learning does not seem to appear in the game industry as part of the finished product, however some games have the option to play versus third party machine learning opponents such as in real time strategy games and traditional games like chess (Justesen et al., 2019).

There are also plenty of examples of machine learning being trained to play games as the player, however letting machine learning fully create the enemy behaviors in video games seem to be lacking research. The reason for this might be that all the information and data that has been written about the traditional methods such as finite state machines and behavior trees (Yannakakis & Togelius, 2018, p. 32). This study aims to start exploring machine learning methods and analyze if there is any viability of this method when creating enemy behaviors.

It has been proven possible to create challenging AI with ML, such as with the 'AlphaStar' project. However it is not a method that is widely used when creating enemies in video games, which has not been widely discussed as for the reason why that is the case.

If games are designed in a way that requires more challenging and/or human-like enemies or advanced behaviors, machine learning methods might be able to create more advanced behaviors compared to traditional methods, which also might be able to adapt to complex game rules. This could allow more advanced games to be designed as machine learning could perhaps simplify the process of creating NPCs.

"-the successful integration of AI in the design process is likely to guarantee satisfactory outcomes for the playing experience. (Yannakakis & Togelius, 2018, p13)

The underlying issue is that there is a lack of research on using machine learning methods to create behaviors for NPCs.

1.3 Research Question

As machine learning is viable in multiple areas within computer science, it does however not seem to be the case for creating NPC behaviors in video games. The purpose of this study will be to explore the viability of implementing machine learning into NPCs and understand the reason behind its underuse among commercial games.

The research question is as follows:

> *How viable is machine learning for creating believable enemies in video games?*

1.3.1 Limitations

- This study will focus exclusively on machine learning for a 3D game with stealth elements, as we need the test environment to have primitive mechanics and game rules. Machine learning AI in this experiment is not comparable to many other game genres, such as RTS games (Yannakakis & Togelius, 2018).
- The results of this study are unlikely to produce similar and comparable results for other game genres as they differ in complexity and game mechanics. For instance if this experiment was done as an RTS or a racing game, the results would be different.
- This study will not focus on how Unity ML-Agents work and perform in comparison to other toolkits.

2 Extended Background

Using machine learning methods to create behaviors for non-player characters is not a well researched topic. However there are many examples of machine learning methods being used to simulate players. Such as the general video game AI, which attempted to create a generalized AI that could finish a variety of games with different game rules without knowing what games were going to be played (Perez-Liebana et al., 2019). Since it is possible to use machine learning methods to simulate human players in video games, it should be possible to create enemies with this method as well (Justesen et al., 2019). However using machine learning to create enemy behaviors seems to be rare, it was not possible to find studies specifically about enemies. Although machine learning has been used as adversaries in some games, we were not able to find a paper attempting to analyze the viability of the process and the believability of enemies that were created with machine learning.

There are numerous projects that focus on training machine learning AI to control 'playable' characters as opposed to non-player characters, which turned out with interesting results. Such is the case for the previously mentioned *AlphaStar* and *OpenAI Five*. These were projects that took years of development and training the AI to an adequate level for showcasing. One thing to note is that these AIs were trained to play as efficiently as possible with disregard to what is considered believable behavior by an implied player, as they are using character entities in a competitive setting which would normally be controlled by actual players. One of the most notable things about such AI when they were showcased on, for example, the *OpenAI Five Arena* (2019) stream on the 'twitch.tv' streaming platform, is that they are incredibly hard to play against. As such a difficulty can be promising as an in-game feature for hardcore players, it may be difficult to restrict the machine learning AI to a casual level of difficulty.

They are also trained as players and play with the same rules and not trained to serve a role as an enemy NPC in a video game. These AIs are not meant to be believable agents of video games, they are only meant to be as effective as possible.

In the paper Game AI Revisited (Yannakakis, 2012, p. 289), it discusses how newer AI methods have been very successful in improving games; it is possible that machine learning could be useful when creating NPC behavior and improve the quality of NPCs. One issue with traditional AI methods is that it is difficult to create behaviors that can adapt to varied environments in the game. As machine learning is not widely used when creating enemy behaviors, it could be possible that machine learning is useful in this regard. Machine learning could learn to understand different environments, how to manipulate and take advantage of the nuanced properties of it which may create more believable agents.

"As AI has already provided satisfactory solutions to most NPC tasks (including navigation and lower levels of NPC control) the focus of research on NPC AI may shift towards under-researched, yet very promising, directions that will enhance NPC capabilities." (Yannakakis, 2012, p289).

"So far, the question of whether empirical research efforts should be put more on the agent or its environment (or both) in order for the agent to appear more believable, human-like, or intelligent remains largely unanswered." (ibid)

2.1 Non-Player Characters

NPCs are essentially character entities in a computer game that are controlled by the computer itself instead of a real player, and they come in many different forms in many different genres, which appearance-wise can vary from the most human-like to the most unhuman-like.
(Warpefelt, 2016, p. 31).

2.1.1 Types of NPCs

With the concept of an NPC, it goes without saying that an "Enemy" is not the only type of NPC that exists for many games. Most commonly in role-playing games (RPGs), there are many different types of NPCs that can be narrowed down to four abstract categories:

- Ally or Friend
  - NPCs that actively support the player to overcome challenges (Warpefelt, 2016).
- Mob
  - Short for 'mobile', a term used in video games which refers to a moving character that can either be an aggressive enemy or a character that is neutral but can be killed for one purpose or another (Hecht, 2007).
- Neutral
  - NPCs that are neither allies or enemies, they may provide service or interaction to some extent but they do not actively help nor pose a threat to the player. Neutral NPCs can include vendors, quest givers, sidekicks and storytellers (Warpefelt, 2016).
- Enemy
  - NPCs that challenge the player. In many cases enemies provide the main challenge in video games (ibid).

2.1.2 NPCs in different genres

There are instances where a character can transition from one category to another based on the game's design. This is particularly evident in games which has a subgenre called 'choices matter', where choices laid out for the player can come in the form of actions or dialogues which are often used as a means to control "relationships" with NPCs, an example would be that an allied NPC turns into an enemy if the relationship value has decreased to a specific threshold.

In RPG the NPCs have many possible roles, therefore there needs to be many types of behaviors programmed. For instance the game might have a vendor NPC, allied NPCs and different enemies that might have different abilities, an enemy dragon NPC would have different behaviors compared to a humanoid merchant type (Warpefelt, 2016).

In RTS games, the AI often controls many smaller units simultaneously. In order to work towards a determined strategy, often the AI has to be programmed with a lot of randomisation in order to make its strategies less predictable (Warpefelt, 2016). In some games like Civilization, the AI has to cheat at higher difficulties because the developers were unable to create the difficulty by improving the enemy AI behavior (Yannakakis & Togelius, 2018, p. 94).

2.1.3 Patrolling NPCs

Patrolling NPCs described by Warpefelt (2016) are "guards" which are a type of NPCs that patrol an area. These NPCs are often designed to walk around an area in search for the player or to defend a specific point, and would react in some way when the player is detected, in some games these NPCs starts attacking the player in order to make the player lose and in other games the player might lose immediately when spotted by this NPC type.

2.2 Concept of enemy AI

Most video games have enemies that have static behaviors with sometimes some randomisation to create a sense of believability. If AI in games are made too predictable for the player, the ‘challenge’ might be easy and the predictability could break the player's immersion. One example of this is enemy patrolling routes, often in games enemies patrol a certain area and repeat the same process on a specific path with waypoints. This can seem very rigid and not particularly believable, however it might also be good for the player to be able to predict the enemy’s path, this depends on what type of challenge the game is intended to have. ML could solve this by having the characters search and patrol in more creative ways that might not be as predictable. The exact purpose of enemy NPCs are different depending on the design of the game and the type of game that the developers want to create, therefore the AI requirements of the enemies are often vastly different depending on the game.

Creating enemies that behave in a believable way is difficult, at the same time the enemies also need to be challenging. One of the biggest issues with creating enemies is their predictability, if the player can figure out what specific variables trigger different AI states in a finite state machine (FSM) or behavior tree (BT), then the player can play the game to basically control the behavior of the AI enemies. To counteract this the developer can create some randomized behavior for the enemies. However the enemies are not really able to challenge a human's intellect and creativity. ML might be able to create more dynamic behavior so that the enemies can use more intelligent and creative tactics and strategies to challenge the player. If ML methods become more easily available, then we might see more believable behaviors from these NPCs.

2.3 Believability

Believability is a term to describe how the player is immersed and engaged in the video game by convincing the player that the game world is believable. It is difficult to precisely determine what believability is and how to achieve a believable NPC behavior (Warpefelt, 2016).

NPC AI has to be designed in a way that makes their behavior seem fitting in the game world in order to achieve believability. An example would be to deliberately make an NPC shoot projectiles less accurately as opposed to giving them the ability to score a hit ever time which can disrupt the fun the player is experiencing, this would also simulate the nature of human error which makes the NPC feel less robotic and more believable (ibid).

“-if humans are expected to play against or cooperate with AI-based bots in video games, other factors come into play. Instead of creating a bot that plays perfectly, in this context it becomes more important that the bot is believable and is fun to play against, with similar idiosyncrasies we expect from a human player” (Justesen et al., 2019, p. 15)

The term believability is synonymous to realistic or expected behavior, and behaviors carried out that “makes sense” to the player. However believability does not mean that the game has to be realistic, just that the game world makes sense by being consistent in terms of the way it looks and plays, the game has to feel right which is difficult to define (Togelius et al., 2013).

The purpose of an enemy is to challenge the player, in regards to believability the enemy has to challenge the player in a way that feels immersive, reasonable and makes sense for that type of NPC. A regular enemy soldier in a video game for instance should not have perfect aim, however an enemy that is portrayed as an expert sharpshooter should have next to perfect aim and be very difficult to challenge.

“NPCs of this type will generally attack the player on sight, and in some cases they will even seek out the player to attack them” (Warpefelt, 2016) describing the Enemy NPC type.

The AI type that will be used in this study will simply search for the player and attack. A combination of the “guard” and “enemy” NPC types which is common in video games.

Loyall (1997) describes how different types of actions an agent does is related to their believability which this study will adhere as believability criteria in order to measure it using the following points:

- Concurrent Pursuit of Goals and Parallel Action

  "Some activities are truly performed in parallel while others are done concurrently with a mix of interleaving of steps and parallel or overlapping action. If we want our agents to have convincing behavior, they also need to be able to perform multiple actions and pursue multiple higher level activities concurrently."

- Appearance of Goals

  "All characters in the arts and nearly all creatures in the world appear to have goals. If we want our agents to be as believable, they need to also appear to have goals."

- Reactive and Responsive

  "Characters in the arts are reactive to changes in their world. It is hard to imagine a character that could be believable that never reacted to a speeding car about to hit him, someone yelling his name, or someone opening the door for him as he was reaching for the handle."

  "These reactions must be at speeds that are reasonable. Even if everything else is perfect, a character would not be believable if its responses were always delayed by a minute. One could also imagine breakdowns in believability if the responses were too fast. The responsiveness of the agent must be within the ranges people are willing to accept as believable"

This study will consider the aspects of believability about the agent's pursuit of the goal of capturing the player as well as how the agent attempts to achieve this goal and its reactive and responsive behavior in its current situation.

Livingstone (2006) describes an alternative but similar criteria for assessing believability, this study discusses and analyzes believability based on these ideas, while the criteria that is used is described above.

	AI should…
Plan	(1.1) demonstrate some degree of strategic/tactical planning (1.2) be able to coordinate actions with player/other AI (1.3) not repeatedly attempt a previous, failed, plan or action
Act	(2.1) act with human-like reaction times and abilities
React	(3.1) react to players' presence and actions appropriately (3.2) react to changes in their local environment (3.3) react to presence of foes and allies
Notable exceptions	1. Might not apply where design/plot calls for impulsive or stupid characters, nor for animals 2. Might not apply where design/plot calls for characters with significantly superior or inferior abilities 3. Might not apply where game-design/plot call for characters with limited awareness

Table 1: The PAR AI Believability Criteria described by Livingstone (2006, p. 10).

2.4 Viability

The center of this study is the viability of machine learning when creating enemy AI for video games. The viability of the machine learning method for video game NPCs will be determined by analyzing these points:

- The effort to set up training for agents compared to traditional method
- How adaptive the agents' behaviors are to changes in rules
- The possibility of changing undesired behavior
- The believability criteria for NPCs
    - Concurrent Pursuit of Goals and Parallel Action
    - Appearance of Goals
    - Reactive and Responsive

The method can essentially be considered viable if these points can be fulfilled with adequate results. As in order to create believable NPCs, said NPCs should be easy to implement, modify and maintain in order to achieve believable behavior in a timely manner.

2.5 Unity game engine

Unity is a 2D/3D game engine that enables developers to create video games and simulations for desktop, mobile and consoles in a 3D environment using the C# coding language together with the 'Mono' framework (*Unity Games Solutions*, n.d.). This game engine is one of the most documented and used game engines, therefore there are alot of tools, packages and information available.

The core concept of the unity game engine are 'game objects', which are functional entities in game environments. Everything that is placed in the game such as the characters, lights, cameras etc. is considered a Game object. And these are very easy to modify through the engine's interface which includes their variables such as coordinates and sizes or whatever script that is attached to the object.

The 'Rigidbody' plays a key role for the movement of in-game characters. It is a physics solution that was created by Unity developers. It handles velocity such as gravity and movement through a 3D coordinate system(x,y,z) modified by time. This allows game developers to quickly implement physics into their game for characters and objects alike which is otherwise a lengthy process to do from scratch. This study will be using the rigidbody component for character movement.

'Unity packages' mentioned in this thesis are extra features that can be downloaded. These include the 'NavMesh'-system, which is used in this study to create pathfinding for the traditional enemy which allows them to calculate efficient paths towards a destination. To create the machine learning agents the experiment uses the 'ML-Agents Toolkit' package, which uses PyTorch (ibid).

2.6 Machine learning

In the context of video games, machine learning is used to train and improve agents to become good at a certain task. This task could be used to debug, create behaviors for characters, procedurally generated content etc.

2.6.1 Reinforcement learning

This method gives the agents rewards for completing the correct task in the game environment, this reward helps the agent to understand that it needs to repeat the actions because it leads to a 'reward'. Negative rewards called 'penalties' are often used in conjunction, this will allow the agent to understand which actions to avoid to reduce the chances of receiving penalties (Yannakakis & Togelius, 2018, p71-72). This is the core method of training the AI for this study.

2.6.2 Proximal Policy Optimization

There are a few different algorithms that are used for machine learning with reinforcement learning, however, 'proximal policy optimization'(PPO) is currently one of the best algorithms to use in order to get the agent to reach positive rewards fairly quickly at the same time as not being as advanced as other algorithms such as the 'Actor Critic with Experience Replay'(ACER) (OpenAI, 2017). PPO algorithm is used with the ML-agents package in this study.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\ ]$$

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

Figure 1: OpenAI PPO algorithm (OpenAI, 2017)

2.7 Traditional video game AI

In the context of this study we will define non-machine learning AI as traditional AI, this is not an established definition as machine learning methods have been used to a smaller capacity in games for a long time such as in ‘Black & White’ where the player’s creature used reinforcement learning. However machine learning is not used to the degree which is done in this experiment where the machine learning enemy is given a large degree of freedom of its actions.

2.7.1 Behavior tree

A behavior tree (BT) model transitions between tasks or behaviors. The behavior tree can have more complex variables to change between tasks. It is difficult to create dynamic behavior with this method as it requires the programmer to design the BT considering all types of events that are likely to occur and to therefore change behavior of the affected agent whenever the game conditions change. A BT allows for many types of variables to change the behavior (often called selector) this can be useful in order to create complex switches between different tasks. BTs are modular so you can use a lot of extra logic, this can be useful for creating more dynamic behaviors, however it is still not able to create fully unpredictable and dynamic behaviors. Halo 2 and Bioshock are two examples of games using BT methods for enemies (Yannakakis & Togelius, 2018).

2.7.2 Finite state machine

A finite state machine (FSM) is a method in which the agent has a finite number of different states and only one state can be active at a time; it transitions between states depending on specified conditions. These states control the behavior of the character, for instance a state machine could have states for running, walking, and jumping. Only one state can be active at a time, so the character can not be running at the same time as walking. States change based on conditions, for instance if an AI enemy sees the player it might switch to an attacking state instead of a passive state. Finite state machines can be very predictable, if the player can figure out what states cause a state transition, the player can control the AIs behavior. FSM can be created with fuzzy logic in order to randomize the behavior a bit, ensuring that the AI is not completely predictable (Yannakakis & Togelius, 2018).

3 Research methods

In order to answer the research question "*How viable is machine learning for creating believable enemies in video games?"*, research methods were combined in order to test the viability and ability of machine learning to create believable agents. In the paper *Assessing Believability* (Togelius et al., 2013), three methods of measuring believability are discussed. First by player experience, which can be measured through observers looking at players responding to the agents via body, head and facial motions, which are then recorded as data. The second method is by logging statistical data in order to measure Believability. The third is a playtest or observations of a game, where testers are answering a survey or interview. A combination is also possible (Togelius et al., 2013).

This study gathers statistical data and attempts to observe the believability based on criteria described in 2.3 through observations and data from logs. During the process of training the machine learning AI and then comparing with the traditional, we conducted observations of their behaviors and evaluated them based on the criterias. These observations are only regarding their movements, so essentially the quality of how they move and what decisions the AI makes are observed.

A game was created in Unity to test machine learning methods and conduct the study. How the game was constructed is described in the next chapter.

The believability criteria, based on *Believable Agents: Building Interactive Personalities* (Loyall, 1997).

- Concurrent Pursuit of Goals and Parallel Action
- Appearance of Goals
- Reactive and Responsive

Alternative criteria is also described in table 1, which was discussed in *Turing's Test and Believable AI in Games* (Livingstone, 2006).

To answer the research question, one experiment was conducted and one simulation:

- During the experiment, the player speed was changed and data of how the efficiency of both the traditional AI and the machine learning agents was collected via logs.
- During the simulation, the quality of the believability of the machine learning agent based on the criteria described.

The observation data is important to summarize the overall viability of the trained AI in both believability and quality in terms of game design. The statistical data is needed to see the amount of consistency of the ML and FSM successfully handling and punishing a moving player character for aimlessly moving around and disregarding enemy presence.

Simulation uses the observation data collection method.

Experiment uses the logging as data collection and the analysis was done with the quantitative data analysis method.

3.1 Experiment

Experiments are done to understand cause and effect relationships (Johannesson & Perjons, 2021, p. 40). In this case the experiment is to test the viability of the machine learning process, by analyzing if the agent can handle changes in the game rules. This test is about the efficiency of the agents affected by changes in game rules (Johannesson & Perjons, 2021, p. 41).

If a commercial video game often requires new updates, if something about the game is changed, the machine learning behaviors should be affected negatively. The experiment was done by changing the player speed variable in order to see how the different agents adapt to changes in the game rules. Since the machine learning agent is trained specifically at a particular player speed, it may not be able to behave as intended if the game has changed certain variables. If the machine learning agents are way worse if the game rules are changed, it might mean that a developer has to retrain the machine learning AI after every update, which could be a time consuming effort.

- Independent variable: Player speed
- Dependent variable: Efficiency of the agent measured by its win rate
- Experiment hypothesis: The Machine learning enemy agent performs worse if the player speed is increased.

3.2 Quantitative data analysis

This analysis focuses on how both ML and FSM agents can handle punishing a player character for wandering aimlessly around the scene at different speeds, the data gathered will be used to find differences in terms of consistency and efficiency of behaviors needed to punish the player character. The data comes from logs and will be analyzed based on this study's definitions of viability (Johannesson & Perjons, 2021, p. 61).

Both types of agents are given the means to succeed as long as correct behaviors are made within a certain time period for each episode of the game, otherwise it is considered a loss. If an agent cannot consistently punish the player for not trying to avoid it, it would mean there is a likelihood that the agent makes illogical behaviors that would cost the player experience.

Statistical data

To export data, CSV files were used collected via C# TextWriter Class in Unity. The data that was collected is the following.

- Win and losses each episode
- Episode time, how long walltime until enemy won
- Episode count

The results could prove useful for video game developers when deciding how to implement AI characters such as enemies, comparing standard traditional AI:s with machine learning methods. Also the results could be useful for other research using machine learning for character behavior.

3.3 Simulation

This method simulates what a real situation would look like, in this case we will simulate AI behavior in a video game (Johannesson & Perjons, 2021), which will be constructed using the 'Unity' game engine. This study automated the simulated player character's movements.

The simulation was conducted after the machine learning AI was fully trained. There were two rooms, both had a computer controlled player in it. One of the rooms had the traditional enemy AI and another had the machine learning enemy AI. The simulation was done by having the game run until both rooms had completed 500 sessions respectively. The simulation is done to allow observation of involved agents. During this process, they were observed in order to understand the believability of the agents, the observations were done based on the believability criteria that is described in chapter 2.3.

3.4 Observation

The ML agents in this experiment was observed in order to gain insight into their learning process as well as their final performance, the research question rely on the learning process and the final result (Johannesson & Perjons, 2021) as the relevant factors for developing ML agents includes time investment, complexity, believability, difficulty etc., which are essential in game development and player experience.

In normal cases, the observation method refers to the observation of real individuals called 'participants' to gather data. This study however will use this method to observe the digital entities of the experiment. Studying their believability is crucial to the study as the goal of developing NPCs does not center around being as 'effective' as possible but rather reasonable and fair for the purpose of better player experience (Skarupke, 2020) (Justesen et al., 2019, p. 15). It is not uncommon for ML agents to miss the middle ground between 'too easy', 'too hard', and 'too weird' which is required to create a meaningful challenge that makes a game fun to play (ibid).

A disadvantage to the observation method is that the data observed in this experiment is to a large degree based on subjective observations that are influenced by the observers personal experiences, what is deemed viable and believable may be different for other observers (Johannesson & Perjons, 2021, p. 59).

During the simulation, the comparison between machine learning NPCs and traditional NPCs was done by observations followed by discussing their behaviors which became the results of this study. Whenever a strange behavior was observed and could be seen repeated several times it was written down. The observations were done during a 2 hour period based on the believability criteria which is discussed in 2.3. The quality of the behaviors was then discussed in order to reach a consensus based on the believability criteria.

3.5 Alternative strategies

An alternative research strategy is to have testers play the game against these trained AIs, comparing their believability and then answering a survey, the results would then be based on player experiences in a playtest. However since this test has very basic graphics and no animations, it may be difficult for testers to assess believability, especially since this study is measuring the believability of the movements of the agents and it might be difficult to do so for testers. According to Livingstone (2006), the testers need to have an understanding of AI in order to produce useful results in terms of believability.

"It is necessary to recognize that people who play few games have such limited experience of game AI that we cannot expect them to sensibly evaluate different versions of an AI." (Livingstone, 2006, p. 11).

"The feedback from novices, while interesting and potentially useful, is less useful for evaluating the AI of a game." (ibid).

Planning a test like this and then finding testers with adequate knowledge on this topic would be difficult.

Another strategy could be a similar one that was used in the paper *Assessing Believability* (Togelius et al., 2013), a group observed different video game AIs that simulated players playing Super Mario Bros and then they assessed their believability. Since believability is subjective, it may have been better to use a similar strategy for this study. However the time spent creating the game and the different AIs, hindered planning and organizing a similar research procedure as was done in *Assessing Believability*.

3.6 Ethical issues

This study is in some way training a computer to use violence in a virtual environment. Therefore a far-fetched concern could be that the methods for training machine learning agents to use violence in 3D virtual environments might be transferable and used for violent applications in the future (Yannakakis & Togelius, 2018).

4 Creating the game

The game used the Unity ML-Agents toolkit in order to create the machine learning agent. The study involved a simulated computer controlled “player” that has basic behaviors, which will try to avoid the enemy as the enemy learns how to stop the simulated computer controlled player. The enemy will use ML to learn new methods in stopping the player.

The reason why the study requires the player to be simulated by an AI is to simplify when training the enemy character(s) with ML instead of manually controlling the player characters during each iteration, which is a repeated process. This study uses many rooms simultaneously in order to speed up the machine learning process, this is a feature in the ML agent package.

The Unity game engine (*Unity Games Solutions*, n.d.) is an environment that allows the construction of video games and other interactive simulations, coupled with many essential scripts and tools that involve physics, logic, feedback, interaction etc. Instead of having the developers creating these functions themselves, it saves a lot of time to use a pre-built engine. Currently, Unity is one of the most popular and documented game engines that are available.

The Unity ML-Agents is a toolkit used for implementing machine learning into a Unity project, which comes with tools to feed the training algorithm with data necessary for the AI to progressively learn inside the simulated environment.

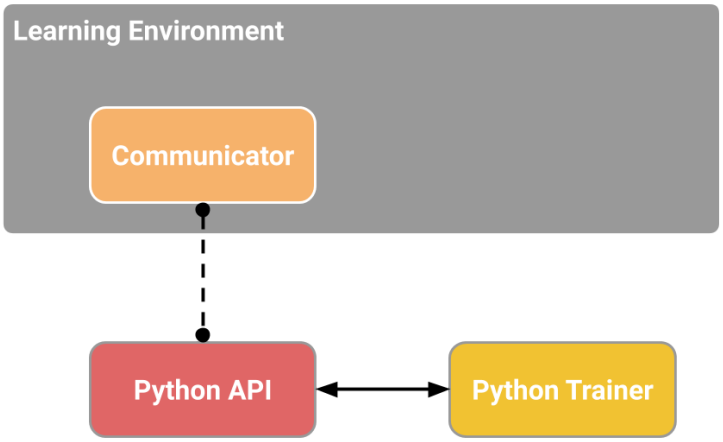


Figure 2: Relationship between python API and the learning environment

4.1 Development process

4.1.1 Unity version and packages

The game is constructed using the ‘2020.3.28f1’ version of the Unity engine, and it is classified as a long-term support version, which means it is more stable than the newly updated versions and will be available for a longer time.

The agents will be trained using the ‘2.0.1’ version (Nov 08, 2021) of the ‘Unity ML Agents’ toolkit. Since the toolkit is an ongoing project, certain aspects of it may be subject to change in later versions which can affect the agent’s performance and learning process.

The FSM variant of the enemy uses the ‘2019.4.0f1’ version of the ‘NavMeshComponents’ package which is for pathfinding in order to generate the shortest path in the environment from one position to another.

4.1.2 Constructing the Scene

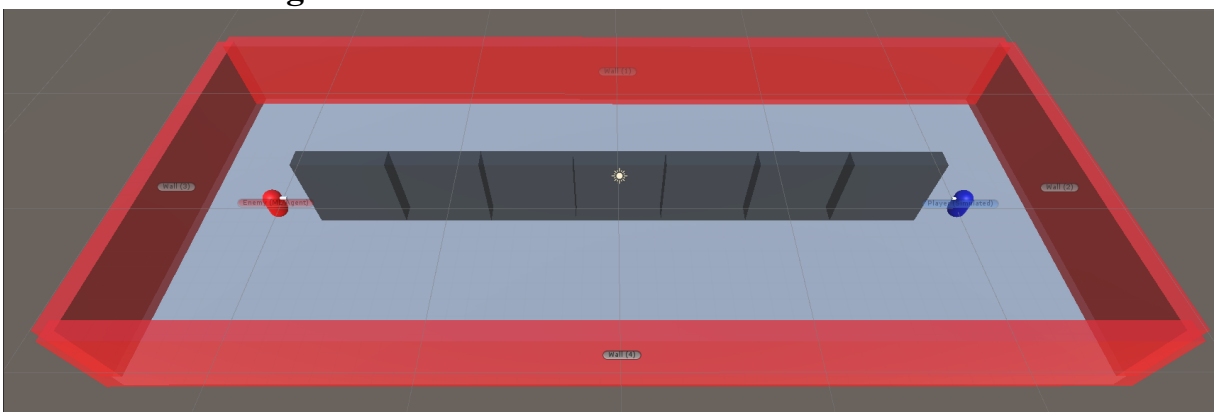


Figure 3: The constructed scene

As shown in figure 3, the scene will be constructed using a simple platform with surrounding red walls. It will also include smaller gray walls which randomize in both position and rotation in order for agents to learn how to react to varieties instead of being reliant on a single level. The two actively moving entities, ‘Player’(blue object to the right) and ‘Enemy’(red object to the left), are placed on opposite sides.

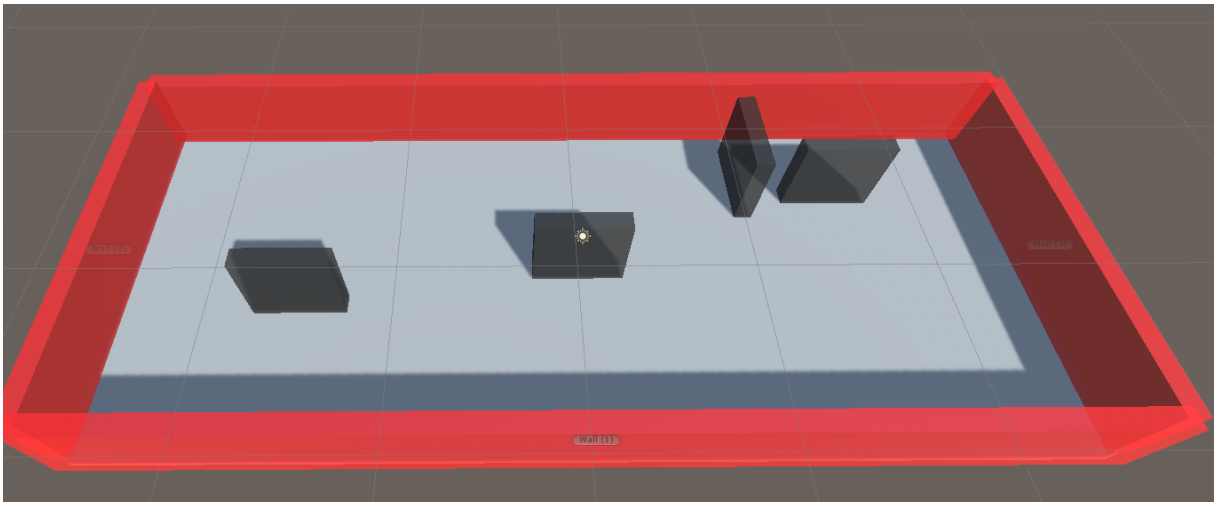


Figure 4: First example of a randomized room

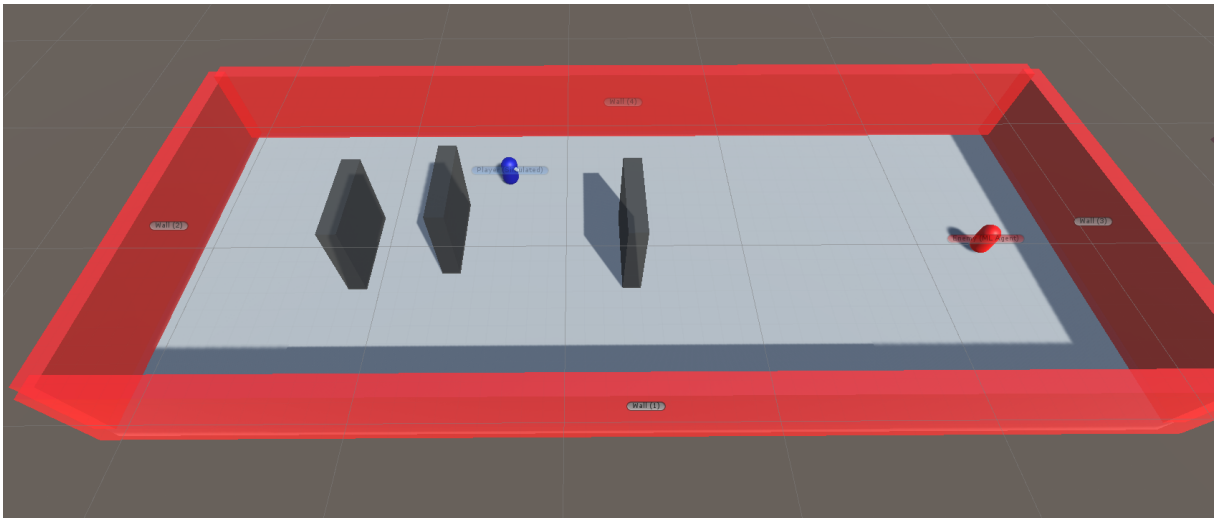


Figure 5: Second example of a randomized room

The walls inside these rooms will randomly be changed each time a game round ends, this is to ensure that the agent learns dynamic behavior so that it can be used in different types of rooms and not just one type of room. The positions and rotations of these walls are randomly changed, this ensures that the enemy agent has to rely more on sensing the walls and the player to adapt to different types of rooms as opposed to following specific paths.

4.1.3 Creating the machine learning agent with reinforcement learning

The machine learning agent has complete freedom to play the game, it has control over its own movement inputs (x,z) and during the learning process it has to learn everything from pathfinding to tactics to defeat the player. It did not know how to play the game at all, they only had rewards and penalties to guide itself to understanding what to do.

The agent also makes use of MLAgent toolkit's 'RayPerception' class which casts multiple detection rays to sense and identify objects to tell wall objects and player characters apart as means to observe the environment and make input decisions based on it. This means that the agent does not know where the player is and has to use its vision with raycasts in order to find the player. To summarize, machine learning agents have simulated vision with the ML-Agents RayPerception class.

When applying reinforcement learning, the highest reward will be +1 and the lowest reward is -1. In the ML-Agents package rewards and penalties are the same variable, a positive reward becomes a reinforcing reward which will make the agent attempt to reach again and a negative reward becomes a penalty which the agent will try to avoid. Rewards and penalties are the same variable, whenever this value is positive it counts as a reward and whenever this value is negative it counts as a penalty. The ML-Agents documentation recommends values between -1 and +1 as rewards.

The enemy receives -0.5 as a penalty for not reaching the player during the episode and it also receives penalties from colliding with walls (-0.0005f per frame) which is capped at -0.5 for a total -1 as a maximum penalty.

The enemy receives + 0.3 as a reward for reaching within 3 units (Unity measurement) and + 0.2 for reaching within 6 units, in combination the enemy can get +0.5 as rewards by getting close to the player. To receive the final reward the enemy must finish the episode by colliding with the player and it then receives +0.5. The purpose of this is to guide the agent into the right type of actions.

Unity ML-Agents uses Proximal Policy Optimization (PPO) algorithm for its reinforcement learning, which is the method that this experiment will therefore use.

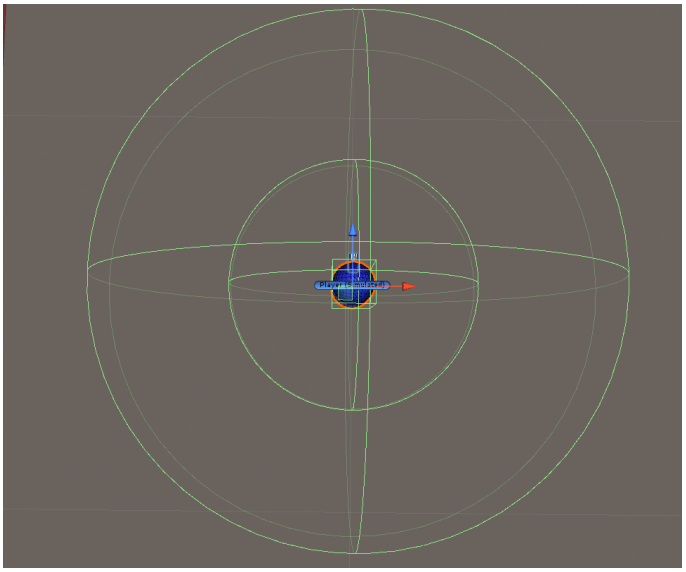


Figure 6: The player character’s colliders

In figure 6, the player character is attached with a set of three colliders, the first and inner box collider triggers a win for the enemy if it is collided with by the enemy. The two outer spheres represent triggers that reward the enemy agent for getting close to the player by intersecting them.

4.1.4 Creating the traditional agent with FSM

The behavior script was created with a simple finite state machine, which includes idling, patrolling, and chasing states. The transition between these states are based on certain conditions. Whenever the player is spotted by a raycast, the enemy will transition to the chasing state and move towards the player in the shortest available path until it loses sight or defeats the player. If the agent loses sight it will go back to the patrolling state from where the player was last seen. The idling state is the default state when the agent is supposed to be idle for a short period of time after reaching a patrolling destination before setting a new random destination. The agent uses the Unity ‘Navmesh’ system to move and utilize pathfinding.

4.1.5 Simulating a player

The player character is simulated in this experiment in order to train the agent and perform the comparison experiment later. This player is a traditional AI type that will randomly decide a direction whenever it hits a wall or does not collide with anything for an amount of time in order to increase unpredictability. The player AI will avoid the enemy and change direction if it sees the enemy in front of it with its raycast vision.

The player agent behaves way more randomly in order to simulate the unpredictability of a player's behavior caused by consequential conditions such as indecisiveness and lack of game knowledge, which should be punishable by the enemy NPC. The study required a simple AI that could easily be defeated in order to train the machine learning agent, therefore the player only avoids the enemy when it sees the enemy in front of it.

4.2 Game rules for the experiment

The experiment will take place in a simple game with movement, collision and simple stealth mechanics, the agents have a vision which is constructed with the Unity raycasts. The goal of the player is to survive for a certain period of time while the enemy agent will attempt to search, find and then reach the player by colliding with it before the time runs out. If the agent reaches the simulated player, the enemy will win the episode and the next one will start after resetting the room and once that is done the room will be randomized again, every episode has a different room layout. The enemy has 30 seconds to win each episode, if it is unable to reach the player within 30 seconds the enemy will lose.

During the training process the player will have a speed of 10 while the enemy while a speed of 20, during the comparison stage that is described later the player speed will be increased in order to test the enemies behaviors after a change in game rules.

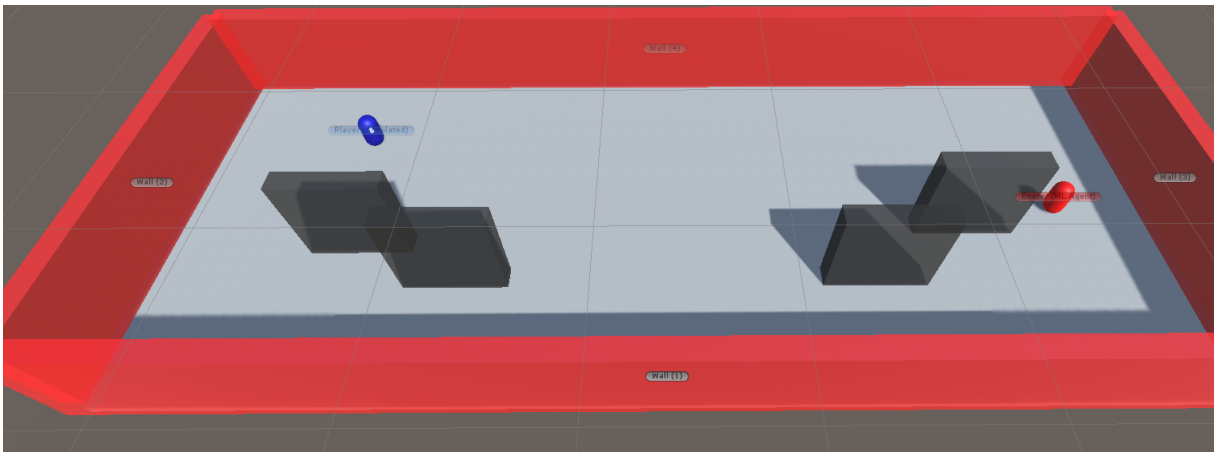


Figure 7: Screenshot of the game, the player(blue object to the left) and the enemy(red object to the right).

Episode definition

Episode refers to a game round, when the enemy wins or loses an episode is completed and the next one begins afterwards.

5 Results

It was possible to create enemy behaviors with machine learning, the machine learning agent was able to win against the simulated player most of the time by patrolling the room and finding the player. The results are based on the following, as described in the research methods chapter.

- Process of creating the agent to analyze the viability
- Experiment involving changing the player speed variable
- Observing the quality of the machine learning agents' behaviors in terms of the believability criteria as described previously

5.1 Development: Training the machine learning agent

When attempting to create a machine learning agent, the game has to be played by the agent in a learning process. During this process the time is sped up by multiplying the scene's time scale variable. The 'Unity ML-Agents' package has the functionality to train the agents by copying and pasting the platform with its agents into the same scene, resulting in multiple instances being run at once, simultaneously feeding the training data. In this study we copied the room 16 times and had the time multiplied by 20, which sped up training a lot while not having any negative effects on the results.

There was a long period during development in which the scripts had to be redone in order to optimize the learning process, during the development we had to run these learning processes at least 100 times in order to observe if it was even possible to create behaviors with the current parameters in the ml agents configuration file and if the rewards and penalty system was implemented in a way to produce useful results.

In order to get the right type of behavior the agent required simple rewards in order to understand that it needed to get closer to the player in order to win, one reward is granted to the agent upon reaching within 6 Unity units and the other one is granted upon reaching within 3 units, full reward is granted when defeating the player. There was also a minor penalty upon colliding with the walls and a penalty for losing the episode.

Time span for the final machine learning process : 1h 20min

Step definition
"Step - Corresponds to an atomic change of the engine that happens between Agent decisions" (Unity-Technologies/ml-Agents, 2017)

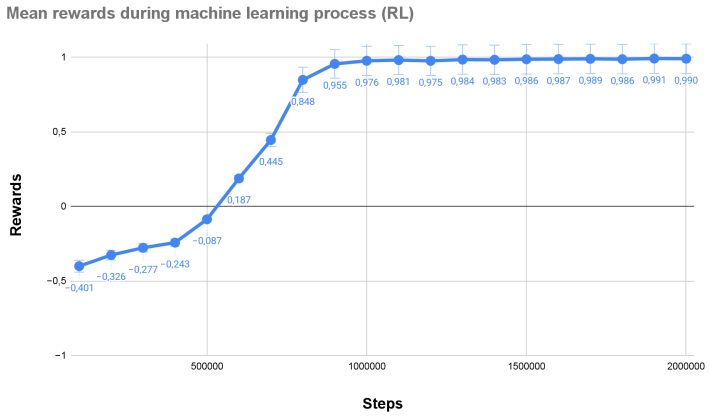
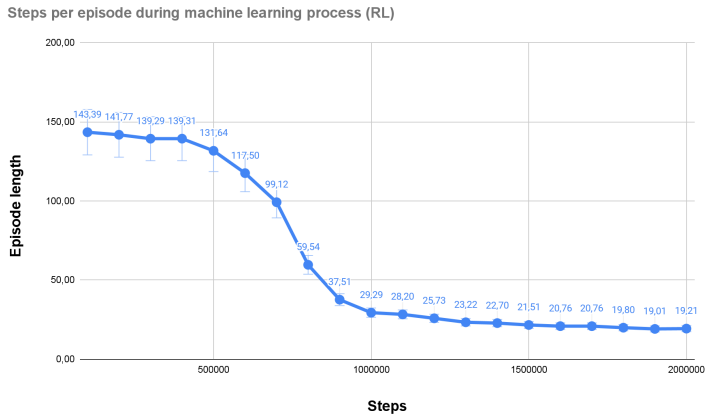


Figure 8: Mean rewards during final learning process



figure

Figure 9: Episode length during the final learning process

5.2 Experiment: Changing the player speed, traditional AI vs machine learning AI

Experiment hypothesis

- The Machine learning enemy agent performs worse if the player speed is increased.

Standard player speed is 10.

A comparison was made by logging 500 episodes of both the traditional and machine learning variants of the enemy agent, an 'episode' is a technical term used for the 'ML-agents' component which is synonymous to a replayable instance. The process was sped up by increasing the time scale variable in Unity. That is why 'walltime' is being used in the following tables, which represents the amount of time elapsed multiplied by the modified time scale. The player was intentionally made to be simplistic in order to effectivise the learning to the machine learning agent, therefore the win rate for these enemies are very high. Another reason for the high win rates is the amount of time for the enemies to catch the players in a fairly small arena.

Machine learning AI

Player speed	Enemy speed	Walltime elapsed to complete 500 episodes (seconds)	Average time per episode (seconds)	Win rate of enemy AI
10	20	1 903.33s (31 min 43 sec)	3.807s	500/500 (100%)
20	20	4 712.43s (78 min 32 sec)	9.425s	465/500 (93%)
30	20	6 125.13s (102 min 5 sec)	12.275s	445/500 (89%)

Table 2: Performance statistics of the machine learning AI agent

Traditional AI

Player speed	Enemy speed	Walltime elapsed to complete 500 episodes	Average time per episode	Win rate of enemy AI
10	20	3 769.92s (62 min 49 sec)	7.540s	500/500 (100%)
20	20	4 511.77s (75 min 11 sec)	9.024s	496/500 (99%)
30	20	5 430.24s (90 min 30 sec)	10.860s	486/500 (97%)

Table 3: Performance statistics of the traditional AI agent

These results show that the traditional AI is highly efficient even when the player speed is changed. At 20 speed the player is equal to the enemy in speed and at 30 speed the player is way faster and should be able to escape, however this AI type still manages to win 97 percent of the time. The machine learning AI is also efficient however, when the game rules are changed by increasing the player speed, it is not as good at catching the player as the traditional AI but still has a high win rate. The machine learning AI was trained with the player having 10 in speed, which means that it is optimized for the player having that particular speed and may have learned certain behaviors that are optimized for a player speed of 10.

When comparing these two results, it seems to suggest that there is some correlation to the hypothesis regarding the increase in player speed would affect the machine learning agent's performance more negatively than the traditional variant. However, the margin is relatively small, meaning that the less effective performance may not be detrimental to its adequacy. The decrease in performance of the machine learning agent is not significant and obviously some decrease in win rate is expected since the player has a higher chance of fleeing if the speed is higher.

5.3 Simulation: Observing the game

5.3.1 Observing the Machine learning AI

Our findings suggest the agent has a tendency to find the easiest and most accessible paths to the area where the player is most likely present based on the walls' randomization pattern. After extensive amounts of training the AI, it can handle traversing around walls in most room layouts but for the rest it would get stuck to a wall on occasion despite being given negative reward for colliding with walls.

Odd and incomprehensive behavior such as erratically moving back, forward, left, and right for seemingly no reason was a constant occurrence for the agent, and it is especially apparent when it tries to chase the player. The erratic movement is not immersive which affects its believability negatively. Its momentum is affected by this behavior, causing it to move slower than the player at equal maximum velocity.

Despite the agent having ray perception that extends for the entire length of the room, it tends not to act accordingly when one or two rays hits the player at a longer distance, this happens randomly and it is unclear as to why this happens. When the agent gets within close proximity to the player, it receives a positive reward to encourage it to get closer to its goal, but it does not seem to realize that it is very close to colliding with the player which gives maximum reward and instead moves away and comes back to finally catch the player in a very inefficient way.

In summary, the agent is fairly capable of tracking the player through remembering patterns but often makes weird behavior that sometimes causes it to lose track of its primary goal in ways that would not be understandable for a real player controlling the player character. The erratic movement and illogical decision making is a common occurrence as it is endlessly trying to test different inputs and patterns. Further training might remove these patterns.

5.3.2 Observing the Traditional FSM AI

The agent had straight forward and consistent behavior with efficient movement which makes them more predictable. The agent's three programmed states: Idle, Patrol, and Chase could be identified visually. Believability of movement could have been improved by adding more randomisation, however this agent does not do obviously strange and weird behaviors.

This agent moves efficiently towards a randomized position and does not make random and sudden deviations, it acts with clear purpose. Whenever the player is spotted, it always acts accordingly and chases the player. It does not give up chasing, unless the player gets lucky and escapes, eventually this enemy type wins almost every time. It acts as intended by the design of the agent and no behavior that is out of the ordinary was observed.

5.4 Aspects that affected the results

The player AI was not implemented with the same complex behaviors as the enemies, therefore the simulated player makes more primitive movements in comparison. It was desired for the enemy to be able to win often in order for the RL rewards to be consistent, otherwise the training would take too long. Since we are not testing these AIs on real players we may have gotten strange results. The issue with attempting training on real players is that it would take too long, during the ML process it took thousands of episodes with 16 rooms simulations at the same time with a time scale of 20, which means that the game was run 20 times faster than the standard time scaling(1).

6 Discussion

The ML agent was difficult to modify and change due to the reward variable being one of the only few factors that can change how the agent behaves. Attempting to change, encourage, or discourage a specific behavior requires re-training its neural network, which may or may not work since it will always try to find the best option to minimize negative penalties and maximize positive rewards.

The agent in this experiment has figured out how to find and reach the player in the fastest possible ways by rewarding it for getting closer to its goal. However, it does not mean it is a fun experience for the player.

It is far from easy to balance negative and positive rewards as there is no clear way of identifying why the AI is making bad or weird behaviors on occasion or when positive rewards drown out the negative rewards. It was even more difficult when attempting to change behaviors that were minor. Setting boundaries for the agent in order to create a more fun or fair rule of play is only possible by discouraging the AI from doing what would naturally bring it closer to its main goal, which can give weird results.

When compared to the FSM agent, the benefit of using traditional methods for creating NPC behavior is that bugs and undesired behavior that may arise are much easier to identify through debugging code compared to machine learning. It is also easier to improve behaviors of traditional NPCs. Traditional methods also give far more control over what behavior should be randomized and how the agent should interact with the environment.

The most notable difference is when setting and changing boundaries for the agent in order to create fair and fun rules of play, an example would be that the AI’s detection range can easily be tuned down to reduce the odds of it sensing the player. Although the ML agent’s ray perception was easy to attach to the enemy agent it is a very complex structure that would not be easy to implement without the ML-agents package. Implementing and modifying perception for the traditional FSM agent is fairly easy compared to creating a simulated vision for a machine learning agent, the ML-agents package has an integrated component (ML-agents RayPerception class), which would be complicated to implement without the package since the information of the rays have to be fed into the machine learning method. Therefore it is not clear how exactly the agents' raycasts interact with the environment and exactly what information it receives from the raycasts.

The results indicate that the ML AI is adapted for the player to have a speed of 10, and when the speed is increased the ML AI win rate goes down, which means that the ML AI is not able to deal with a change in movement speed as well. The ML AI is also way more efficient when the player has 10 speed compared to a higher speed compared to the traditional AI. However the traditional AI is able to get way better and consistent results at all types of speed.

6.1 Viability assessment

6.1.1 Development limitations

The machine learning agent was able to handle other speeds of the player, however the agent's win consistency gets reduced, it is necessary to retrain the agent in order to get better agent behavior after a change in player speed or any other changes to gameplay.

Due to the difficulty of changing how the ML agent should behave in a specific way, it becomes close to impossible to finetune behaviors in order to achieve believability since it would require retraining the AI, whereas in comparison to the FSM agent, it is fairly simple to change the behavior in order to fit the agents requirements of believability. There are some behaviors that the agent occasionally performs that are strange and seem out of place and it is not clear how to improve the rewards and penalties in order to remove these behaviors.

6.1.2 Using machine learning for enemy NPCs

The only way to change the behavior is to retrain the agent with improved rewards and penalties which is very time consuming. Problems with ML behaviors force the developers to expand the rewards and penalties in order to attempt to get better behaviors from the training process, which often creates behaviors that are unpredictable.

The file sizes ended up being large for a small project of this experiment, a larger game with more demanding graphics and programming may cause machine learning behavior to take considerable amounts of disk space while not giving any clear advantages as the modularity of traditional AI methods.

The traditional AI was also a lot more efficient at doing its intended task at different types of player speeds, which seems to indicate that small changes in the game would make the previously trained ML behavior a lot worse, forcing the developers to retrain the ML agent after each update to the game.

In the context of this study, it was not reasonable to use a machine learning agent for this particular game. Creating the traditional AI was simpler, more time efficient and produced behavior that was easily modified.

6.1.3 Fulfillment of believability criterias

Following the believability criteria mentioned by Loyall (1997):

1. Concurrent Pursuit of Goals and Parallel Action
The ML agent is able to patrol and search for the player in reasonable areas of the room, while often reacting if the player is close, although sometimes the agent does not react even though it has clearly spotted the player. The agent seems to be able to patrol and search for the player concurrently.

2. Appearance of Goals
The ML agent does appear to search for the player, however sometimes it ignores the player in easily winnable situations. During the observations, it was clear that the ML agent does not patrol the room in a believable way. Even though it receives penalties from colliding with walls it will still often collide for seemingly no purpose and it will often jiggle side to side and front to back randomly, which looks strange. The agent is sometimes not able to act on obvious information for instance when it is very close to the player and can clearly detect it with its raycasts, it does not seem to attempt to reach the player. Even though the behavior has a lot of issues it will in most cases eventually start behaving more effectively during an episode and then win.

3. Reactive and Responsive

The ML agent occasionally reacts to walls blocking its path, but it is not consistent and sometimes the agent is stuck in the wall for the rest of the episode and loses. There are cases where the enemy agent behaves in a believable way, it will occasionally look around corners and chase the player as intended but this is not a consistent behavior, the enemy will usually start behaving effectively at some point in the episode, but before it does it can act in these strange manners as described. Further training or improvements in the rewards might make this behavior more desirable.

6.2 Development and processing effort

Creating ML AI took several weeks to first learn the ML agents package then testing multiple learning processes in simple games in order to understand how to design the scripts for the ML agent including the rewards, penalties and what type of inputs the agent has access to. ML-agents AI requires more disk space, currently the ML behavior saved is 336KB vs traditional AI script is 6KB. Using ML agents requires the ML Agents package which is currently about 41MB in size and the required python files which are needed for training are in total 4.15 GB in disk space, these can be removed after the training.

One advantage with the ML process was that the AI only needed two inputs to function and not much coding had to be done, however the code did require a lot of rework because the ML agent needs a well designed rewards and penalty system in order to learn properly (Skarupke, 2020).

Comparison of enemy creation process

Approximate effort to set up Machine Learning agent

- Difficult to set up machine learning training, took 1 week of testing and troubleshooting the rewards system.
- Long training process of approximately 1 hour and 20 minutes.

Approximate effort to set up Traditional agent

- Implementation has taken approximately 4 hours.

6.3 Conclusion

It is possible to use machine learning methods for creating enemy AI, however the behavior of the agent in this experiment is not fully believable for an implied player. The agent had freedom to play the game in any way possible and it was able to learn pathfinding methods on its own as well as tactics to defeat the player, although sometimes the behaviors are not fully believable. It is still definitely possible to use this method to create enemy behaviors.

Creating machine learning agents with more restrictive behaviors may solve the believability issue, compared to letting the agent do its actions freely and randomly as was done in this experiment. It is therefore possible that a more restrictive method could be a better option when attempting to create a machine learning enemy. Although if the agent is too restrictive there might not be any purpose of using this method at all, since the advantages of this method is the full automation and randomness of creating behaviors. It would probably be better in that case to use traditional methods, since in that case there might not be any point in attempting this method for NPC behaviors. Perhaps using only smaller aspects of machine learning could create some interesting results, behavior that are mostly created with traditional AI methods but minor machine learning methods such as reinforcement learning could potentially be added to smaller functions in order to perhaps improve a traditional AI. On the other hand it is questionable if this would produce any behaviors that would otherwise be simple to create with traditional methods, which would defeat the purpose of using machine learning.

With all advantages and disadvantages of utilizing machine learning for NPCs considered, the unpredictable nature of machine learning proves the difficulty of shaping desirable behaviors. Machine learning could therefore be considered as being too risky of a venture to implement into commercial games for enemy behaviors. As designing behaviors for NPCs is a key component to making NPCs believable, unless it is intentionally added by design where its advantages outweighs the disadvantages. In other game genres such as RTS games, this method may be more advantageous to attempt, future research could explore machine learning in other genres.

The maintainability of ML agents is very challenging to manage as changing specific behaviors requires training the agent further to adapt to said changes which puts other already existing behaviors at risk of change. As machine learning behaviors are far too complex to identify and debug, it will lead to an indeterminable amount of time and effort to create an agent of adequate quality.

The file size of machine learning AI scales drastically in comparison to traditional AI script files even for a simple NPC concept, it may prove to be an obstacle for a game of much larger scale, but for smaller and compact games it may be manageable. Although this is unpredictable, as it is not possible to know how big the files will get in order to get the desired results.

It is difficult to modify the behavior of ML agents and it seems to be unlikely that machine learning methods will create enemy behaviors that a game requires. However, further developments in machine learning toolkits may make these methods more viable in the future. In conclusion, the effort required in order to make these behaviors at the current state of machine learning are probably not worth the time investment when making a game. Machine learning NPCs are unpredictable and minor behavior problems are close to impossible to fix, which makes this method at this time mostly impractical for commercial games as opposed to viable.

7 References

Arulkumaran, K., Cully, A., & Togelius, J. (2019, July 13). Alphastar: An evolutionary computation perspective. In *Proceedings of the genetic and evolutionary computation conference companion* (pp. 314-315).

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d.O., Raiman, J., Salimans, T., Schlatter, J., … Zhang, S. (2019, December 13). *Dota 2 with large scale deep reinforcement learning*. arXiv. https://doi.org/10.48550/arXiv.1912.06680

Fürnkranz, J. (2007). *Recent advances in machine learning and game playing*. ÖGAI Journal.

Hecht, E. (2007, Feb 20). *The complete WoW abbreviations* [Archived Jun 31, 2009]. Retrieved Apr 26, 2022, from https://web.archive.org/web/20090831112100/http://www.wow.com/2007/02/20/the-compleat-wow-abbreviations

Hu, J., Niu, H., Carrasco, J., Lennox, B., & Arvin, F. (2020, Oct 29). Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning. In *IEEE Transactions on Vehicular Technology* (Vol. 69, Issue 12, pp. 14413-14423). https://doi.org/10.1109/TVT.2020.3034800

Johannesson, P., & Perjons, E. (2021). *An Introduction to Design Science* (2nd ed.). Springer International Publishing. https://doi.org/10.1007/978-3-030-78132-3

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Herper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2020, May 6). *Unity: A General Platform for Intelligent Agents*. arXiv. https://doi.org/10.48550/arXiv.1809.02627

Justesen, N., Bontrager, P., Togelius, J., Risi, S., IT University of Copenhagen, Copenhagen, & New York University, New York. (2019). *Deep Learning for Video Game Playing*.

Livingstone, D. (2006). *Turing's Test and Believable AI in Games*. https://dl.acm.org/doi/abs/10.1145/1111293.1111303

Loyall, A. B. (1997, May). *Believable Agents: Building Interactive Personalities*. School of Computer Science Computer Science Department Carnegie Mellon University Pittsburgh. https://apps.dtic.mil/sti/citations/ADA327862

OpenAI. (2017, July 20). *Proximal Policy Optimization*. OpenAI. Retrieved May 5, 2022, from https://openai.com/blog/openai-baselines-ppo/

OpenAI Five Arena. (2019, April 21). Twitch. Retrieved June 17, 2022, from https://www.twitch.tv/videos/414642196

Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J., & Lucas, S. M. (2019). *General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms*. arXiv. https://doi.org/10.48550/arXiv.1802.10363

Riitahuhta, A., Sudweeks, F., & Gero, J. S. (Eds.). (2012). *Artificial Intelligence in Design '96*. Springer Netherlands. https://doi.org/10.1007%2F978-94-009-0279-4_9

Skarupke, M. (2020, January 29). *Why Video Game AI does not Use Machine Learning*. Probably Dance. Retrieved April 25, 2022, from https://probablydance.com/2020/01/29/why-video-game-ai-does-not-use-machine-learning/

Togelius, J., Yannakakis, G. N., Karakovskiy, S., & Shaker, N. (2013). Assessing believability. In *Believable bots* (pp. 215-230). Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-32323-2_9

Unity Games Solutions. (n.d.). Unity. Retrieved April 25, 2022, from https://unity.com/solutions/game

Unity-Technologies/ml-agents. (2017, Sep 19). GitHub. Retrieved June 18, 2022, from https://github.com/Unity-Technologies/ml-agents

Warpefelt, H. (2016, May). *The Non-Player Character: Exploring the believability of NPC presentation and behavior*. ResearchGate. https://www.researchgate.net/publication/303496966_The_Non-Player_Character_Exploring_the_believability_of_NPC_presentation_and_behavior

Yannakakis, G. N. (2012, May). Game AI revisited. In *Proceedings of the 9th conference on Computing Frontiers* (pp. 285-292). https://doi.org/10.1145/2212908.2212954

Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games* (Vol. 2, pp. 2475-1502). Springer International Publishing. https://doi.org/10.1007/978-3-319-63519-4

Appendix A - Figures showing

RayPerception functions

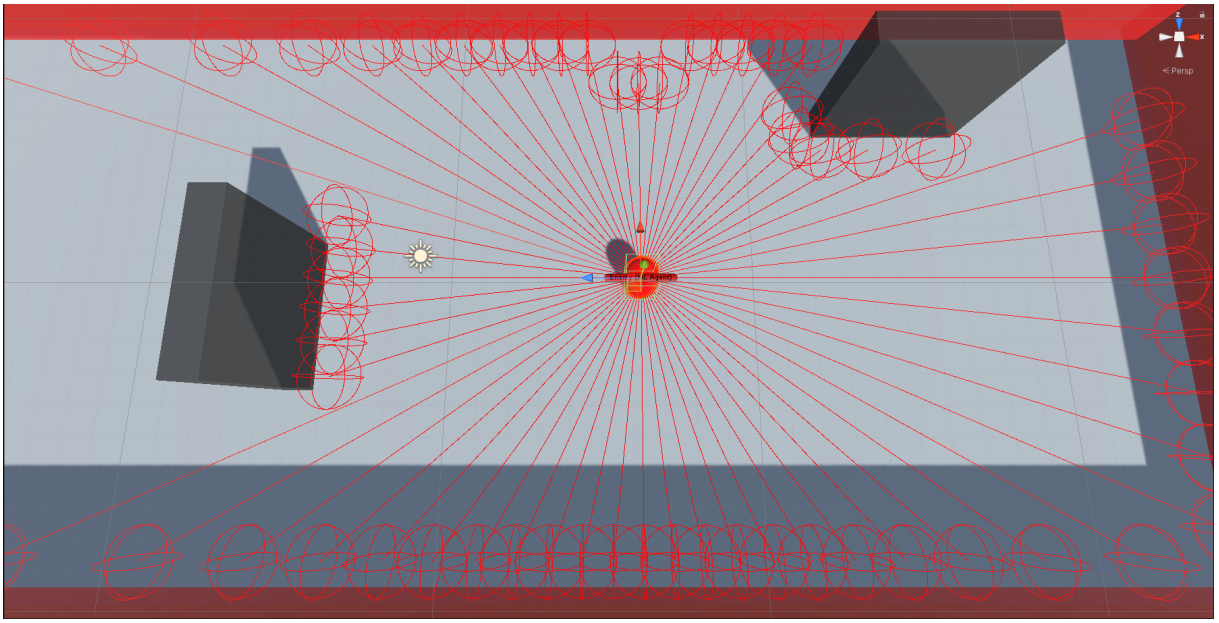


Figure 10: Showing the ML agent RayPerception class (vision simulation)

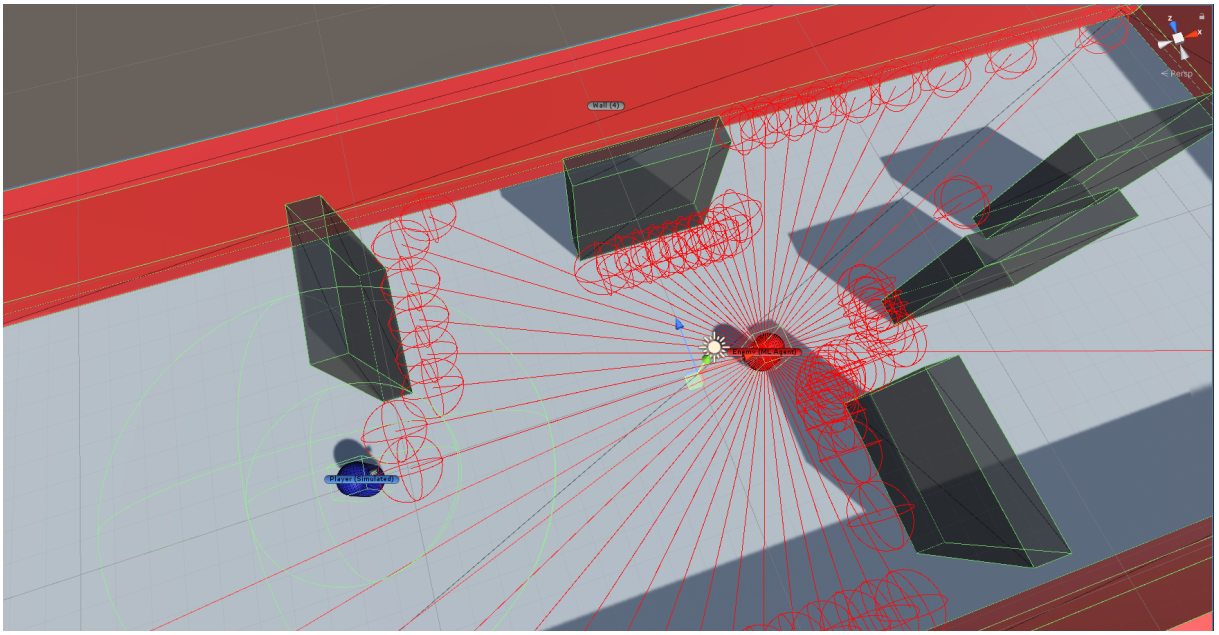


Figure 11: Showing the ML agent RayPerception raycasts colliding with the player

Appendix B - Notes from observation

Viability observations for the machine learning agent

Incredibly difficult to maintain the agent
Inability to debug and change undesired behavior for machine learning NPCs
More time consuming development effort
Enemy could become too difficult and potentially not be fun to play against for some players
Difficulty in designing behaviors that are created by machine learning AI, unlikely that the AI will get intended behaviors
Machine learning requires more space for the files on disk, if the NPC behavior is more advanced for a game with a larger scope, the file sizes might be very large. This might make the games performance worse

Table 4: Notes during observation

Appendix C - Unity ML-Agent hyperparameters


```
default_settings: null
behaviors:
  Enemy:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.01
      epsilon: 0.3
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
    init_path: null
    keep_checkpoints: 5
    checkpoint_interval: 50000
    max_steps: 2000000
    time_horizon: 64
    summary_freq: 100000
    threaded: false
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: null
  env_args: null
  base_port: 5005
  num_envs: 1
  num_areas: 1
  seed: -1
  max_lifetime_restarts: 10
  restarts_rate_limit_n: 1
  restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 20
  target_frame_rate: -1
  capture_frame_rate: 60
  no_graphics: false
environment_parameters: null
checkpoint_settings:
  run_id: test
  initialize_from: null
  load_model: false
  resume: false
  force: false
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

Appendix D - Graphs from machine learning training process

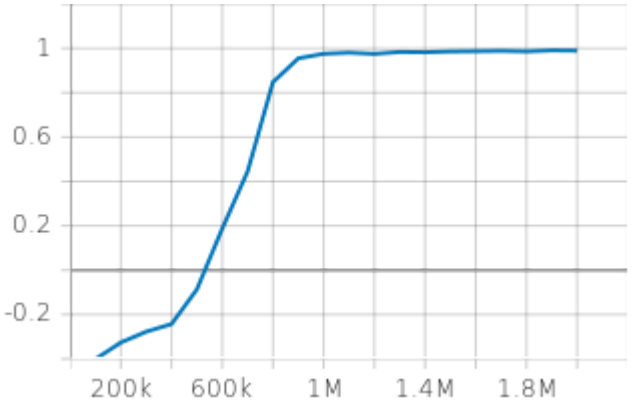


Figure 12: Environment Cumulative Reward

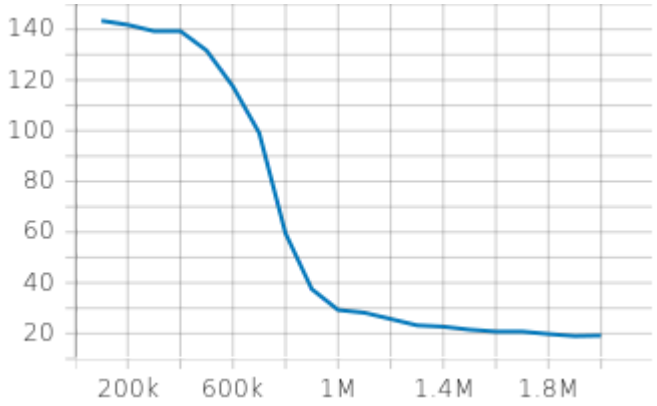


Figure 13: Environment Episode Length

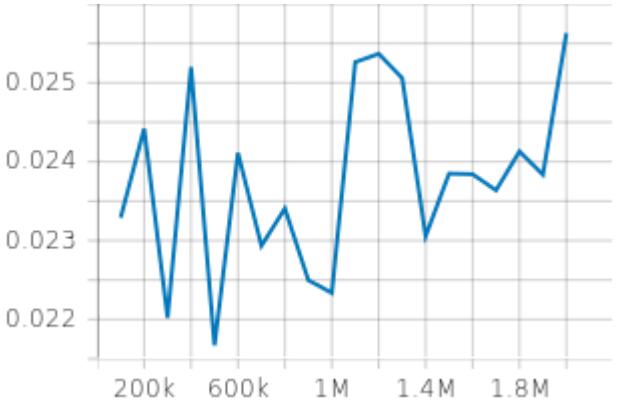


Figure 14: Losses Policy Loss

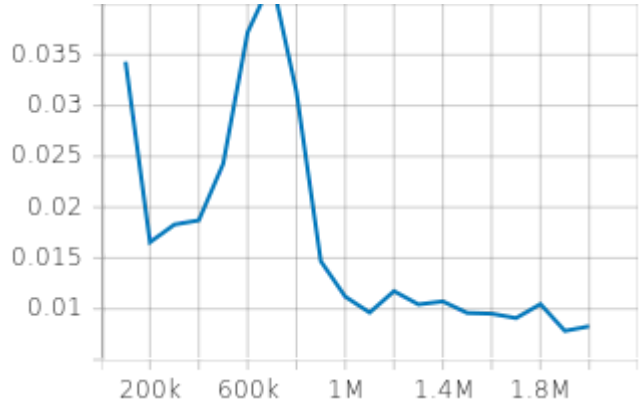


Figure 15: Losses Value Loss

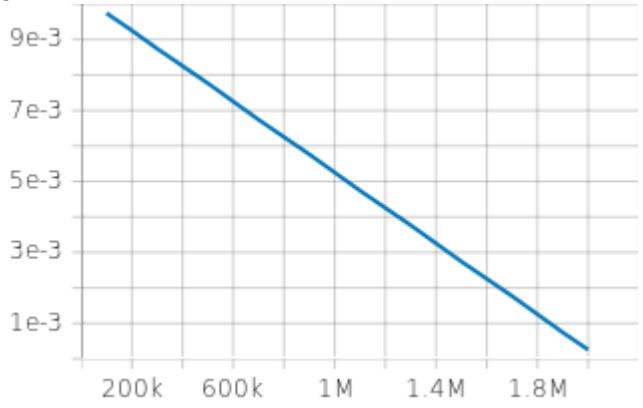


Figure 16: Policy Beta

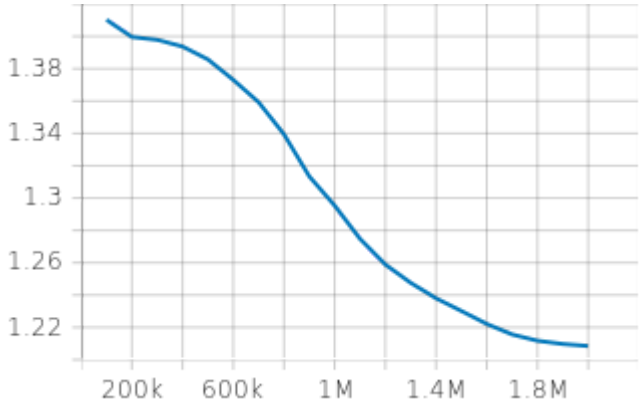


Figure 17: Policy Entropy

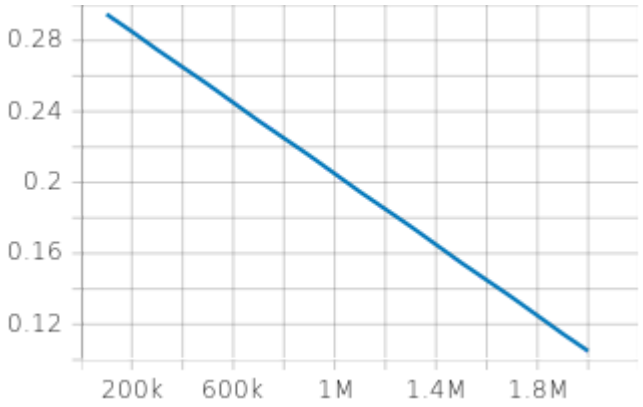


Figure 18: Policy Epsilon

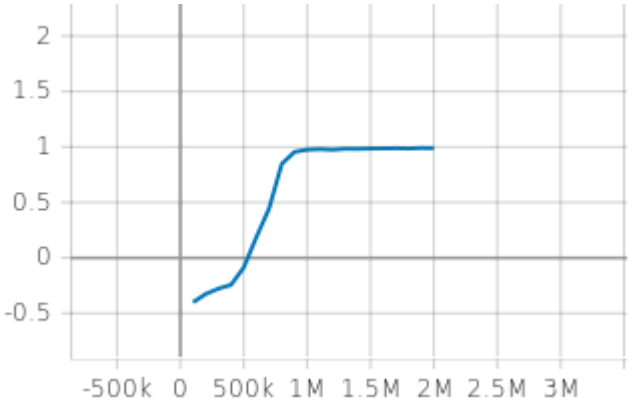


Figure 19: Policy Extrinsic Reward

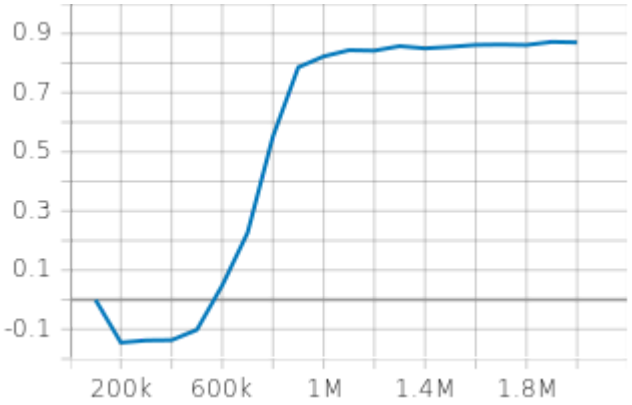


Figure 20: Policy Extrinsic Value Estimate

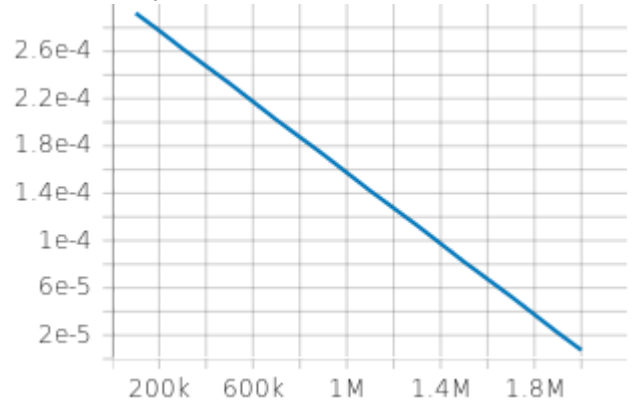


Figure 21: Policy Learning Rate

Appendix E - Reflection Document Mattias Larsson

I think the study has achieved its purpose to investigate how useful machine learning for NPC is in video games. I still think that machine learning is possibly useful for AI in other game genres such as strategy games but it might not be worth the time required to implement. The focus became the study itself rather than going through the criteria of the course, so we adjusted it to fit the course afterwards. The planning was unstructured, but for us i think it worked but we were late for a few unofficial deadlines. Perhaps we should have had a more organized planning, but it ended up working well.

The work is fitting and relevant for Computer Game Development, as artificial intelligence is a very important field for video games, although the definition is a bit different in terms of video games. Even simple behaviors are called artificial intelligence in video games but in computer science in general it is a different process. In some way we are applying the typical definition of artificial intelligence onto video games.

When we discussed the thesis our plan was to create a game and then attempt to use machine learning enemies. We had seen how games such as Dota2 and board games like chess and Go had advanced AIs that used machine learning and were able to beat the best players in those games. Because of that we were curious how useful it was to use machine learning methods to create video game enemies. We thought the best way was to attempt to create the enemies with machine learning and then writing about the process and observing the results. We thought that criterias such as believability was important since we did not necessarily want to create the most challenging AI, just something that seemed to be an enemy.

The machine learning process was difficult, we started attempting to create a simple game and then using machine learning to learn an agent to walk through a room that was randomized, this took probably 50-100 attempts before we started to understand how to design a rewards and penalties system that would work. The documentation for the ML-agents package is not always clear how certain features worked so we had to try things ourselves in order to figure them out. We tried to change the hyperparameters often but we were not sure what the documentation meant in regards to certain variables, the descriptions were often unclear. After we had learned how the package worked we again probably tested the learning process 100-200 times with the finalized game rules that is explained in the thesis in order to get the type of behaviors we wanted by adjusting the rewards and penalties. We tried a lot of different methods such as.

- Minor rewards for the agent to walk in the right direction
- Higher penalties for ending the epsiode far away from the player
- Different penalty amounts for colliding with walls
- Also tried no penalties for colliding with walls
- Rewards for seeing the player with raycasts
- Higher rewards if the agent caught the player quickly
- Penalties for staying in the starting zone

By far the most effective method was to give smaller rewards for getting closer and closer to the player, we ended up using this system for the final machine learning agent. By the time we finished the experiment we realized that the machine learning method was way more complicated that just creating an AI with traditional methods. However we did not record this process, which could perhaps have been useful for the thesis.

The most relevant course has probably been the AI Based Experience Design (AIBU) course, which was partly about machine learning AI for video games as well as believability in video games. In the courses such as Project work in Game design (SP:PROJ),Game mechanics (SPM) and Mobile Application Development (MAPP), I mostly did programming and sometimes did npc behaviors for enemies which was helpful when doing this work since I had some background knowledge of the topic.

It was difficult to define exactly what research methods we were going to use, we knew that we wanted to observe the enemies in order to understand their behaviors but the definitions of observation methods did not really match what we wanted to do. So we tried our best to define what we wanted to do.

Perhaps it would have been a better solution to test this game on other players and do some sort of interview or survey, however we did not have the time to set this up since the machine learning process was very time consuming.

Studying NPC behaviors should be relevant in the future if I end up working in game development, perhaps what I have learned with machine learning is applicable to other fields as well. I am pleased about the work and I think we tried to explore this subject in a reasonable way.

Appendix F - Reflection Document William Örnquist

Introduction

In this thesis we analyzed a method for creating artificial intelligence (AI) called ‘machine learning’, focusing on non-player characters (NPCs). In which, we gathered information about the principles of creating NPCs and the current state of machine learning. We would then proceed to experiment the use of machine learning as the main component for NPCs’ decisions and actions, comparing side-by-side with NPCs using traditional AI.

The conclusion was made based on the results that machine learning is currently in a state that is not suitable to meet the fundamental requirements to create quality NPCs.

After researching the subject of creating machine learning AI for NPCs, I have learned the reasoning behind why machine learning in commercial video games is exceedingly rare and unpopular regardless of how much of a standard it became in other fields within computer science. I have also gained the knowledge of how implementing machine learning into NPCs work in practice and what its strengths as well as weaknesses are based on my prior experience with more traditional methods of creating AI.

Before the Research

I was well aware of the existence of machine learning AI being experimented on controlling the player characters of existing games as a third-party tool. The demonstrations of big projects such as OpenAI, AlphaGo, and AlphaStar were such examples I have witnessed and was fascinated by. The only question I had at the time was why this advanced technology has yet to be taken advantage of in commercial games as far as NPCs are concerned.

What has allowed us to further dig into this topic is the knowledge and experience we gathered from relevant courses such as ‘AI Based Experience Design’ (AIBU) and ‘Game mechanics’ (SPM) where we acquired essential insights on the design and implementation of AI both machine learning and traditional. Along with other courses which involved programming that helped us being able to set up the game environment and script the AI.

During the Research

The progress was steady but unforgiving when we got started with the experiment, during the course of which we had to go through many trials and errors to make the machine learning NPC of our experiment behave properly to the environment by controlling the distribution of ‘rewards’ which is one of the very few variables that we developers have to encourage and discourage certain behaviors, but the result would always be unpredictable which speaks for itself how risky of a venture it would be for a commercial game. We have thus far only been able to find a select few references which specifically entails the subject of the absence of machine learning NPCs in most games, but there is plenty other research which in abstract regards to for example what makes NPCs believable to support the claim that a machine learning NPC that can fulfill its role and goals in some way does not entirely translate to a successful design or better player experience.

As we began to focus on collecting data, we found no means to measure the ‘believability’ of an NPC digitally, hence why the observation method is so important. We instead opted to measure the effectiveness of how both machine learning and traditional NPCs carried out their task, as it would be relevant to their capability of punishing a randomly walking player character not adhering to dangers. Which can be used in the argument of believability.

Conclusion

The research has given me an interesting insight on what machine learning lacks in the game projects and what it means to create interesting NPCs, as it also serves as an example that there is no need for NPCs to be perfect as they are many times made deliberately flawed as a design choice for the reason of for example making challenges challenging but beatable by keeping the middle ground between ‘too easy’ and ‘too hard’.

I believe this thesis will serve as an informative ‘heads up’ for future projects about the use of machine learning to take caution on its unpredictable nature. It will also be well understood that at the time of this research, machine learning is still in its experimental phase and has been an area in the game industry that has been through years of development and research to get this far and still has a lot of room for improvement and potential to better simulate NPCs.