Final Master Thesis

MASTER'S DEGREE IN AUTOMATIC CONTROL AND ROBOTICS

Implementation of multiple collaborative agents using reinforcement learning

MEMORY

Author :	Filip Torjusen Byberg
Supervisor :	Prof. Dr. Cecilio Angulo
Date :	January, 2024

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Abstract

The aim of this thesis is to explore the possibilities and capabilities of machine learning (ML), more specifically reinforcement learning (RL) within a 3D digital environment. By utilizing the Unity's ML-Agents toolkit, the aim is to showcase the user-friendliness of the implementation and promote the use of the applications, emphasizing simplicity and logic. The environment takes place in Unity, where the tasks consist of training multiple agents within the same scene, to act cooperatively in order to achieve a desired behavior. The project consists of providing behaviour guidelines for the agents and exploring strategies to achieve better performance.

An online visualization of the final product can be found in the following link. This link contains an online server where the project is hosted in the form of an interactive game. Its possible to scroll trough the different scenes of the game by pressing "1", "2", "3", "4" or "5" on your keyboard. Each scene contains a result that is relevant in this thesis. As a bonus, by pressing "5" on your keyboard, a scene where you can control one of the characters exist:
`https://filipbyberg.itch.io/2v2-volleyball-rl-ml-agents-showcase`

ETSEIB

ETSEIB

Contents

ETSEIB

List of Figures

ETSEIB

List of Tables

Acronyms

AI Artificial Intelligence

API Application Programming Interface

BC Behavioral Cloning

CL Cumulative Learning

CR Cumulative Reward

GAIL Generative Adversarial Imitation Learning

ID Identification

LRS Learning Rate Schedule

MA-POCA Multi-Agent POsthumous Credit Assignment

MARL Multi-Agent Reinforcement Learning

MDP Markov Decision Process

ML Machine Learning

NN Neural Network

NPC Non-player Characters

OOB Out Of Bounds

PPO Proximal Policy Optimization

RL Reinforcement Learning

SAC Soft Actor Critic

SDK Software Development Kit

TL Transfer Learning

Chapter 1

Introduction

Imagine you and a friend are on your way to play a game of volleyball together. You find yourselves on a sandy beach, with the sun shining brightly, the sound of birds chirping and a slight breeze of wind coming from the coast. As you set up the volleyball net, you realize that these *observations* might affect your play style, strategy to win and chances to perform well. A cast of wind might alter the balls trajectory, the sun can blur your vision or a loud noise from a bird can distract you. These are all disturbances that might cause a bad outcome for your game, and hinder your ability to focus on observations such as ball trajectory, net height or proper positioning.

While playing the game, consider the multitude of activities available: you could serve the ball to the other side, place yourselves close to the net to potentially block smashes from the opponent, or even try to rapidly move around the court to confuse your opponent. Each of these *actions* has its favorable or unfavorable place in its *environment*, you both must make numerous strategic decisions that bring you closer to a desired objective. Furthermore, you must make these choices while dealing with external disruptions and occasional distractions. How do you manage to convert these observations into actions that enhance your chances of winning in this particular environment?

Crucial to your ability to succeed in this game is the power of associating actions under certain circumstances in the environment, with rewarding or penalizing outcomes. Trying to distract an opponent with fanatic movement instead of playing the game might not be a good strategy in the long run. However, shooting high easily receivable ball trajectories might lead to an easy goal for the opponent, if they have the capabilities and intention to smash the ball. Positioning yourselves correctly is also a factor, as usually the opponent seek to shoot the ball towards somewhere where it's challenging to counter it. Representing specific patterns of play and strategies as a product of your observations and objectives is highly beneficial. Indeed, many fundamental aspects of playing volleyball depend on crafting these patterns; analyzing potential scenarios, adapting based on past experiences and creating long-term plans to deal with certain situations. The ability to learn this is a fundamental skill for any player, whether human or artificial.

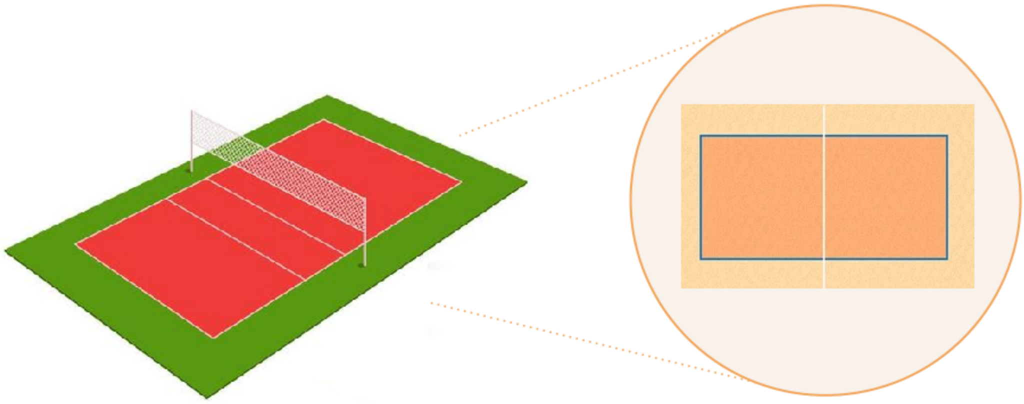


Figure 1.1: Simplifying environment for abstract strategy

In the context of a volleyball match, imagine that you and your teammate create a strategy similar to what is shown in Figure 1.1. Using this simplified model that retains essential information, you can make efficient decisions with less exploration, make more accurate predictions, and create strategies. Ultimately, this approach enhances gameplay and increases the chances of winning. But how do you arrive at this strategy? And how can you teach an Artificial Intelligence (AI) such behaviours through interactions in the environment? These questions are central concepts of Machine Learning (ML) and Reinforcement Learning (RL).

This master thesis explores the concept of RL and its significance in cooperative gameplay. It focuses on the agents learning when given certain behavior criteria and placed in a volleyball court with a teammate and two opponents. The aim is to shed light on strategies for shaping agent behaviour in the context of cooperation and studying observations, rewards and environmental designs impact on the performance.

1.1 Motivation

Machine learning has increasingly been rising in popularity, as its field of use is continuously expanding and is applicable to almost every innovative business. The amount of attention it has received has allowed most of the public to recognize its great potential, especially since the rise in popularity of Large Language Models (LLM) like *ChatGPT*, and non-player characters (NPC). The primary motivation for this project is to contribute to the knowledge of AI behaviour, especially to those in the field of game development.

It is also worth mentioning my personal motivation to gain experience in 3D development platforms, such as Unity. I have always had an interest in video games and 3D scenery. Being able to work with something that is deeply related to my personal interest is exciting. I am extremely grateful that I get the opportunity to work with something that is both relevant and exciting, given the increasing popularity of ML.

1.2 Objectives

The main objective of this master thesis is to research the use of the ML-agents toolkit on the 3D game development platform Unity. An already existing $1v1$ volleyball project will be used as a starting point. By developing further on this project, the aim is to explore different aspects of the toolkit while expanding from a $1v1$ game, to a fully functional cooperative $2v2$ game.

As a start, the agents should be able to take individual decisions in order to clear the ball from one side to the other. However, as the project progress, more complexity is necessary in order to achieve cooperative team behaviour between the agents. Furthermore, the agents should be able to mimic standard strategies such as shooting, passing and strategic positioning. To achieve this, a set of milestones have been designed and are listed below:

1. Research and evaluate
    - ML-agents Toolkit [3]
    - Unity framework [5]
    - Existing volleyball project [4]

2. Design desired behaviour with observation and rewards
    - Shooting over the net
    - Passing to teammate
    - Proper positioning
3. Explore strategies for training efficiency

The goal is to implement this project as an open source code, for other people that are curious about Unity's ML-agents.

1.3 Scope

This master thesis will explore the possibilities of the ML-agents tool-kit, developing further on an already existing project consisting of a simple $1v1$ volleyball game. The platform used will be Unity game development platform, where OpenAI's recent released class of RL algorithm, Proximal Policy Optimization (PPO) will be used because of its ease of use and good performance.

From here, the master thesis will focus on different strategies to implement certain desired behaviours for the agent, with the aim of achieving cooperation between agents. Specifically strategies for shaping rewards, design of environment, choice of observations and achieving robust policies. Finally, different approaches will be compared by analyzing TensorFlow graphs and performance.

1.4 Declaration

The content of this thesis is written by the student, but with the help of writing tools such as *Grammarly* and *ChatGPT* to help correct misspellings and provide suggestions for improving text quality.

All proposals for text improvements from AI tools were personally evaluated, in order to not move away from the personal style of writing and formulation.

Chapter 2

Background

Reinforcement learning (RL) has its origins from animal behaviourism and the study of positive reinforcement by behavioral psychologist B. F. Skinner in the 1930s [6]. Skinner demonstrated that animals could be trained to perform complex tasks through simple reinforcement mechanisms, such as receiving a food reward for performing a desired control. These psychological theories quickly transitioned into the computational models in the 1950s and 1960s. A breakthrough was when Richard Bellman introduced his concept of the Bellman equation, describing the relationship between the value of a state and the values of its successor states, providing a way to compute optimal policies.

Today, RL defines the problem facing agents that learn to make good decisions through action and observation alone. To be effective problem solvers, such agents must efficiently explore their environment, assign credit, and generalize to new experiences, all while making use of limited data, computational resources, and perceptual bandwidth [7]. With this method, a behaviour can be tough to the agent, by rewarding desired actions in certain situations and punishing undesired ones.

2.1 Choosing simulation platform

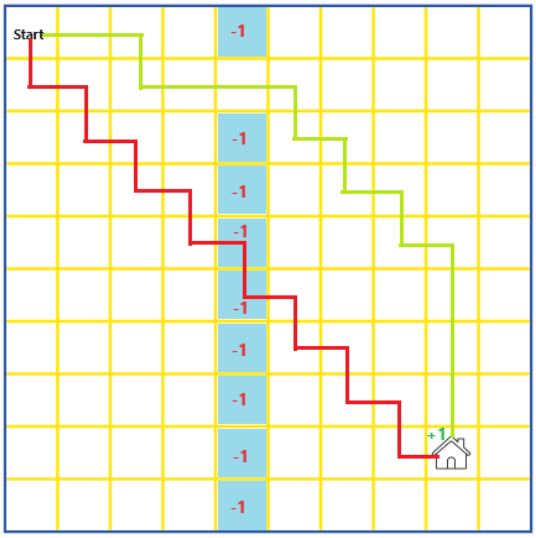
The increasing popularity of AI has brought with it more complex and realistic simulated environments then ever. A common factor among many existing environments has been limited visual capabilities, inaccurate physics or poor interaction among artificial agents. Authors discuss in [3] the importance of simulation platforms in the context of AI and RL research, with a focus on the Unity game engine and the Unity ML-agents Toolkit. A main tribute in the paper include a taxonomy of simulators based on flexibility of environment specification, highlighting a variety of examples for illustrative purposes.

A great addition to the paper is the showcase of ML-agents Toolkit's fourteen example environments [8]. These examples contains various environments for testing and learning ML in Unity. The research conducted consist of comparing PPO and Soft Actor Critic (SAC).

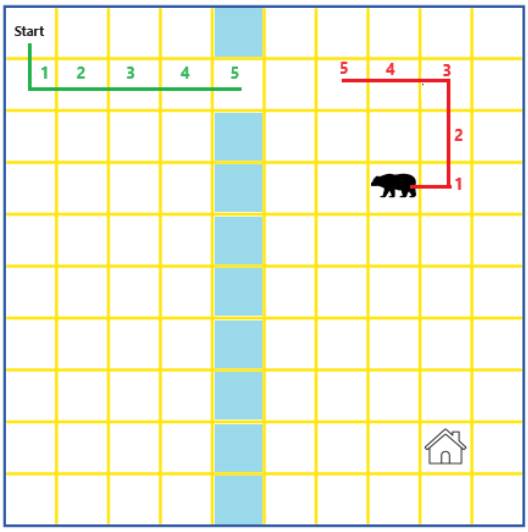
2.2 Reinforcement Learning

To understand the concept of RL, it is important to address the main attributes that together enables the algorithm to work as intended. RL requires the use of an *environment*, as mentioned in Section 1, the environment in this case is the virtual volleyball court in Unity. An environment consists of multiple *states*, which can be either discrete, or continuous as in this environment. The environment serves as an operating space for the free roaming *agent*. The agent is assigned a finite amount of *actions*, which allows it to move around. This can typically be; move forward, to the right, stand still, jump, etc. For every action taken by the agent, it will be assigned to a new state. This enables the algorithm to recognize behavioural patterns, by acknowledging the connection between specific states and actions.

Simply by having an agent with assigned actions within an environment, it's possible to define the agents movements. However, without any objective the agent will wander around with a lack of purpose. In order to give the agent a sense of direction toward a goal or a specific behaviour, *rewards* and *penalties* are introduced. These attributes enables rewarding or penalizing certain *events* that occurs as a consequence of an action or sequence of actions in an arbitrary state. Typically the agents are rewarded for events such as reaching a goal state or any other customized event that will promote a desired behaviour. By assigning a negative reward to the agent, its possible to penalize the agent when an undesired event occurs, leading to the agent steering away from behaviours that relates to such events.



(a) Rewards and penalties



(b) Observations

Figure 2.1: Reinforcement learning example

As an example, let's imagine a simple scenario where the agent needs to navigate its way trough the woods from a starting point to its house. This is depicted in Figure 2.1a with a 10x10 discrete environment, where each square is a state the agent can reach. The objective is to reach the house, so naturally the agent is rewarded for reaching this state. There is however a river flowing trough the woods, and to make sure the agent does not fall in the river a penalty can be applied to the agent upon reaching these states. In order to learn, the agent needs to explore its environment to create a pattern in its behaviour, understanding what actions are beneficial in which scenarios. Without the penalty of stepping into the river, the agent would most likely find a path shown with a red marker in Figure 2.1a, while in reality crossing the bridge as in the green path would be more beneficial.

Even though AI can simulate human behaviour, it lacks the basic senses of a human. Since agents don't have eyes to observe its environment, *observations* can be used to inform about external data from the environment. Let's imagine the same scenario as before, but now a dangerous bear is also moving with random actions in the woods. The penalty of encountering the bear would be large and a reset to the simulation. Shown in figure 2.1b, a scenario where the agent has reached the bridge after five actions takes place. Here the agent risks encountering the bear in the following state if the action of moving to the right is chosen. By giving the agent observations about the position of the bear, allows the agent to take this into consideration when choosing its next action.

2.3 Training for a behaviour

Training an AI can be very frustrating. The training often takes too long and the results are almost always different or worse than expected. In RL terminology, an *agent* is trained to learn the correct sequences of *actions* to take on a given *state* of its environment in order to maximize its *reward* [1]. This can be illustrated as shown in Figure 2.2.

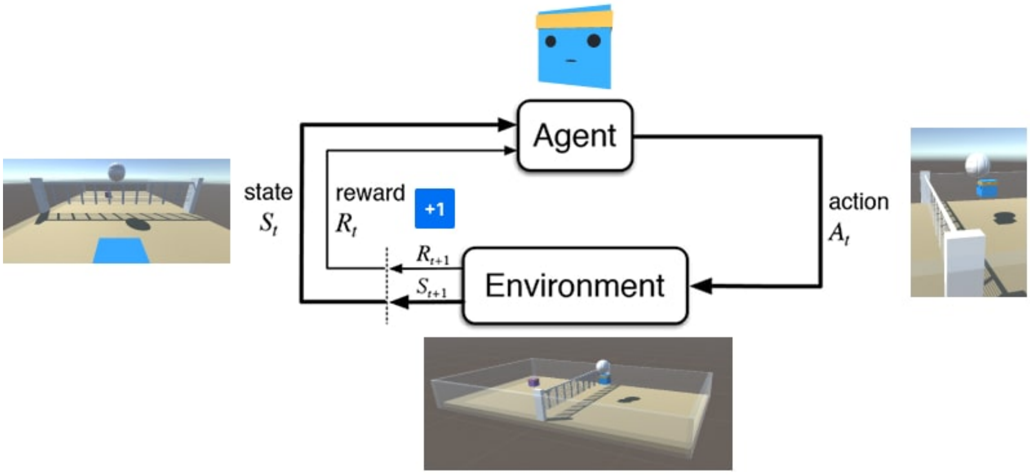


Figure 2.2: Illustration RL terminology. Source:[1]

There are many ways to train an agent, but the behavior can vary depending on which algorithm is chosen. Popular algorithms such as SAC [9], Behavioral Cloning (BC) [10] or Generative Adversarial Imitation Learning (GAIL) [11] have been used in the past to train agent behavior. However, with the emergence of a new class of RL algorithms, PPO released by *OpenAI* in 2017 [12], this algorithm has quickly become one of the top contenders due to its harmonious trade-off among simplicity of implementation and the convenience of parameter adjustment. Its objective is to calculate an update in each iteration that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. PPO uses a novel objective function shown in equation 2.1, that corresponds to the clipping heuristic proposed in [13], which is not typically found in other algorithms:

$$\begin{aligned} L^{CLIP}(\theta) &= \hat{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \\ r_t(\theta) &= \frac{\pi_\theta(a|s)}{\pi_{old}(a|s)} \end{aligned} \tag{2.1}$$

where:

- 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

The policy π has a certain mystique to it, as it can be hard to interpret the meaning of. With formal terminology, the policy is defined in terms of the Markov Decision Process (MDP) [14]. The surrogate objective L uses the advantage estimate $\hat{A}_t(a, s)\ \forall s, a$ to evaluate an action a in a state s, compared to a randomly sampled action from a previous iteration policy $\pi_{old}(\cdot|s)$. Based on these estimates, L is then optimized using multiple epochs of stochastic gradient ascent to find a new parameter vector θ for the new policy π_θ [15]. This procedeure effectively creates a strategy that an agent uses to chose the best next action in pursuit of a goal. Furthermore, no matter which algorithm is chosen, certain guidelines are necessary in order to achieve good results.

2.3.1 Simplicity

The single biggest and most common mistake is to start out too complex. Complex environments are interesting, but creating simple environments and behaviours minimizes the margin for making mistakes and undesired results. When wanting to achieve a certain behaviour for an agent, think about what the simplest version of this problem would be. This could be things such as, excluding unnecessary obstacles, minimize randomness and limit amount of observations. An excessive amount of input parameters can serve as noise or disturbance. It's desired to make sure that the AI can fail quickly and often. Once the agent can reliably reach the goal or act a certain way, the foundation of the environment is working. This opens up the possibility to slowly add complexity to converge towards a desired behaviour.

In Unity and ML-agents, there are many ways to deal with this. One is which is Curricular Learning (CL) [16], which involves training an agent on a sequence of tasks or sub-tasks, gradually increasing in complexity. The agent begins with a relatively simple task and then moves on to more challenging ones.

Another viable approach is transfer learning (TL) [17]. This approach involves reusing knowledge gained from one task to accelerate learning in a related or different task. Instead of a sequential progression, transfer learning aims to transfer skills and experiences learned in one context to another.

2.3.2 Manual testing

When training an agent, it can take thousands, if not millions of steps before the agents actually starts to adapt its behaviour. If this is your method of testing for every single change in the code, it will require a large amount of time. One thing that is necessary for any ML-agent developer is to be your own tester. This can be done by controlling the agent manually to test that all the functionality and rewarding is correctly done. This will ensure that the environment will not break before the training. Luckily, as Unity is a game development platform, adding self-play to a character or agent is an easy task [18].

2.3.3 Reward shaping

Choosing the correct rewards alone can be a quite tedious task, as it impacts the agents task prioritizing the most. Normally, the two most effective ways to achieve a simple behaviour is to assign a goal reward and an existential penalty. This is an effective way to teach the agent to reach a goal and to do it fast. In many cases, when the environment is more complex, it can be tempting to add more rewards with different values. The problem with this is that for every new reward introduced, the model get increasingly more complex. With more different rewards, comes an increased amount of possible *local maxima* in the *cumulative reward* (CR) for the agent [19]. Meaning there will be multiple different behaviour "branches" that the agent can get attached to. It is also apparent that an agents' behaviour tend to converge toward a behaviour "branch" if the rewards are easier accessible. This is commonly known as the *cobra effect* [20].

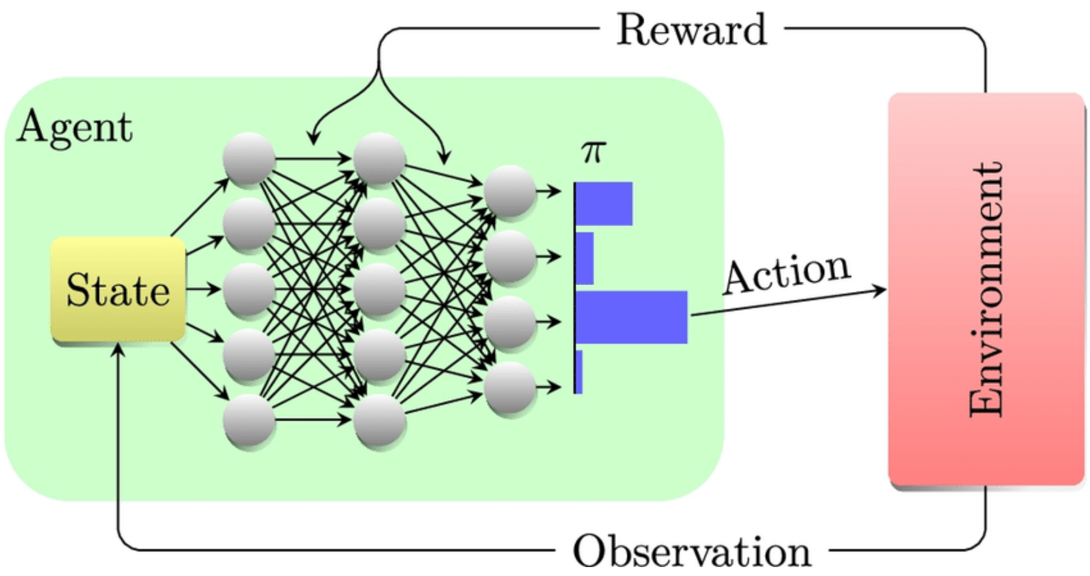


Figure 2.3: Neural Network (NN) in RL. Source: [2]

Complexity hinders the agents ability to learn quickly and as intended. Figure 2.3 shows their attribute to the neural network (NN) model to be trained. As unique rewards and observations are inputs to the model, they can serve as disturbances and contribute to creating a bad policy, if poorly designed. By limiting the amount of inputs to the model, its possible to limit the margin of error, and create a more predictable outcome.

2.4 Multi agent reinforcement learning

In the traditional approaches of reinforcement learning, the task is to train a single model for a specific behaviour for an agent. The problem emerges as soon as multiple agents are introduced. Training a single model with multiple behaviours, depending of the state can be a challenge. It is discussed in [21] the challenges and solutions in the field of Multi-Agent Reinforcement Learning (MARL), specifically focusing on issues related to the early termination of agents within an episode. In this context, a terminated agent can be any of the agents that did not contribute with a certain action in order to receive a reward. Existing MARL algorithms address this by placing a terminated agent in an *absorbing state* [22], which simplifies training, but has efficiency and scalability concerns. The authors propose a novel architecture called Multi-Agent POsthumous Credit Assignment (MA-POCA), which uses attention mechanism instead of the fully connected layers. The attention mechanism attributes future expected value to states with terminated agents without relying on absorbing states.

A popular way to create a functional learning environment in the field of MARL is the use of a shared policy between agents. This requires all agents to make decisions in a decentralized manner to optimize a global objective. In [23], a hierarchically decentralized MARL framework was proposed to enable the agents to dynamically share rewards to encourage cooperation. This concept is further explored in [24], where the PPO algorithm is used for MARL in order to perform well in various challenging video games that require cooperation.

Chapter 3

Resources

This master thesis employs existing and well-tested tools and libraries to further develop a more complex project. These tools allows for a more user-friendly overall experience and are meant to attract and engage newcomers that are interested in learning 3D development and ML. In this chapter, a short overview of the tools, libraries and repositories are presented to give a better overview of their place in the project.

3.1 Unity Game Engine

Unity is a powerful and versatile game engine that has become a big part of the video game development industry [5]. Created by Unity Technologies, it has gained widespread popularity for its user-friendly interface, cross-platform capabilities, and robust feature set. Unity is not only used for game development but also for creating interactive 2D and 3D applications across various platforms, including mobile devices, desktop computers, consoles, and even augmented and virtual reality devices.

Unity is responsible for hosting the 3D environment where the project virtual environment will take place. The reason for using Unity is for many of its key features that comes in handy for this project, such as:

- **Graphics and Rendering:** Unity provides high-quality graphics capabilities, including real-time rendering, post-processing effects, and support for advanced shaders, enabling developers to create visually stunning scenes.

- **Physics and Simulation**: Unity includes a physics engine that facilitates realistic physics simulations for objects and characters within the game world. This feature is crucial for creating lifelike interactions and environments.

- **Scripting:** Unity supports multiple programming languages, including C#, which is the most commonly used language for scripting in Unity. The scripting system is flexible and robust, empowering developers to create game logic, behaviors, and custom functionality.

- **Package Management:** Unity Package Manager enables developers to easily integrate third-party assets and plugins into their Unity projects. These packages can include everything from 3D models and textures to code libraries and tools

3.2 ML-Agents toolkit

ML-Agents toolkit is an open-source project by Unity Technologies that empowers researchers and game developers to integrate ML algorithms and AI into Unity-based games and simulations. This toolkit serves as a bridge between the Unity game engine and various machine learning libraries, making it easier to train agents to perform tasks within virtual environments.

Key features and components of the ML-agents toolkit include:

- **Unity Integration:** The ML-agents toolkit seamlessly integrates with the Unity game engine, allowing developers to create realistic 3D environments and simulations for training AI agents. This integration simplifies the process of connecting AI models with virtual worlds.

- **Customizable Environments:** Users can design and customize environments within Unity, defining the rules, physics, and visual elements of the simulation. This flexibility is valuable for creating diverse and complex training scenarios.

- **Unity ML-agents toolkit Gym:** Gym is a component of the toolkit that enables integration with the OpenAI Gym interface. This allows users to leverage a wide range of reinforcement learning algorithms and benchmark their Unity-based environments against other RL environments [25].

- **Training Options:** ML-agents supports various training algorithms, including PPO, SAC, and more. Users can choose the algorithm that best suits their specific tasks and objectives.

- **Observations and Actions:** Developers can define observations and actions for the AI agents, specifying what information the agents perceive from the environment and how they interact with it. This allows for fine-grained control over the learning process.

- **Unity ML-agents Python API:** The toolkit provides a Python API for users to train, evaluate, and analyze AI agents with TensorFlow graphs.

- **Unity Inference Engine:** ML-agents includes an inference engine that allows trained models to be used in Unity without the need for a separate Python environment. This is crucial for deploying AI-powered agents in real-time applications and games.

3.2.1 ML-agents SDK

The three main components in the ML-agents SDK are Sensors, Agents, and an Academy [3]. The Agent component is used to indicate that a *GameObject* [26] within a scene can function as an agent, be able to collect observations, take actions and receive rewards. The Academy keeps track of the amount of steps in the simulation, manage the agent [27] and enables the ability to define environmental parameters. The sensor component allows the agents to have observations in the scene.

Figure 3.1 shows how the ML-agents SDK allows developers to flexibly train agents in the environment. "Behaviour A" illustrates how two agents can share the same policy, something that will play a critical role in this master thesis. When training, they interact with the Python API or already trained neural network models. It is also possible to link an agent to non-policy scripts or heuristic player inputs.

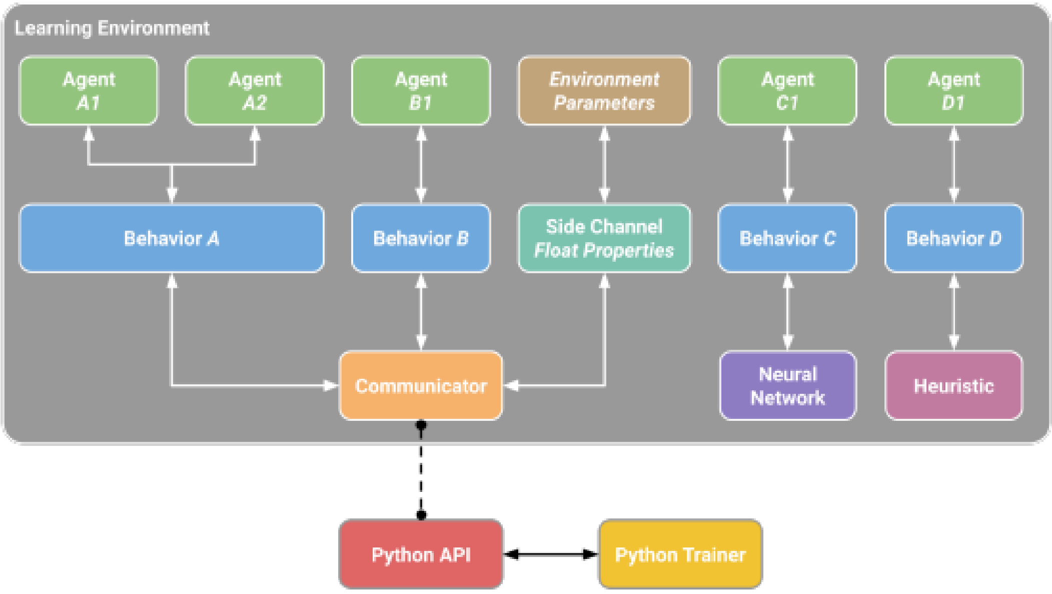


Figure 3.1: Different ways of interacting with an agent in learning environment. Source: [3]

3.3 Ultimate Volleyball project

The *Ultimate Volleyball* project by Joy Zhang [4] serves as the starting point for the project and requires access to all of the mentioned sections in this chapter. This section highlight the most important aspects from this project and the necessary information to understand the starting point for further development.

3.3.1 Environment

The environment as shown in Figure 3.2 simulates a simple volleyball court, consisting of two court sides, a net, surrounding barricades and two agents. However, for this projects purposes, another agent will be added on each team.

The floor consist of a walkable *GameObject* that provides collision physics and a trigger zone on each side that detects when the ball hits the floor. Around the court is an invisible boundary, that resets the scene if the ball falls out of bounds. Another important note is that the agents appear square shaped, but has a sphere collider to help control the ball trajectory.

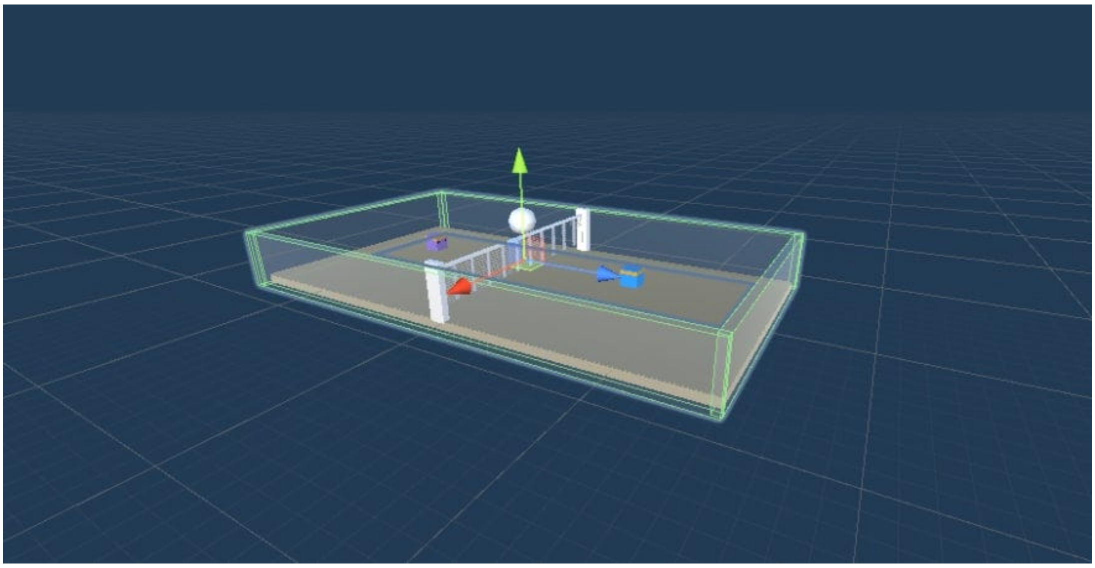


Figure 3.2: 3D Volleyball environment. Source: [4]

3.3.2 Actions

The actions available for the agent in this project includes the following ones:

- 0 / 1 / 2 = No movement / move forward / move backward
- 0 / 1 / 2 = No movement / strafe left / strafe right
- 0 / 1 / 2 = No rotation / rotate clockwise / rotate anti-clockwise
- 0 / 1 = No jump / jump

The agents will only know these actions as encoded integers and in the beginning of a training session, every agent will start by selecting actions at random. These actions will be kept throughout the project, as more actions is unnecessary.

3.3.3 Observations and rewards

The observations of the agent is how the agent sees it environment. In ML-agents, three types of observations can be used, such as:

- Vectors [28]
- Raycasts [29]
- Visual/camera input [30]

In the *Ultimate Volleyball* project, 11 vector observations were used:

- Agent's y-rotation (1)
- Agent's x, y, z-velocity (3)
- Agent's x, y, z-normalized vector to the ball (3)
- Agent's magnitude distance to ball (1)
- Ball's x, y, z-velocity (3)

Furthermore, the rewarding of the agents was kept very simple, +1 reward each time an agent hits the ball over the net.

Chapter 4

Implementation

This chapter aims to explain how a solution was implemented for the multi-player case. It will detail the process of designing a new environment tailored to a $2v2$ play-style, including the rationale behind the selection of observations. Additionally, two main approaches, the first one includes the integration of two distinct models, one for each agent on the team, while the other approach to create a shared policy for both agents. To facilitate this, a reward shaping strategy will be presented, offering a structured approach to effectively encourage behaviors during various stages of the training process, ensuring consistent progress toward achieving the desired final behavior.

4.1 Environment design

To properly train any NN model with RL, one must always start by designing a proper environment. As mentioned, the 3D volleyball environment will be used, but extending the project to a $2v2$ game requires adding an additional agent on each team. Since the environment is already prepared for enabling the shooting behaviour, it is necessary to enable the new desired behaviours for correct positioning and passing.

Training one model to teach both agents cooperative play may result in undue complexity, while training two separate models may lead to less cooperative ability. The aim is to use a role-based approach, with each agent assigned a specific role, as a result of a shared policy or two unique policies for the individual role. As illustrated in Figure 4.1, one agent will be responsible for the offensive role, involving shooting the ball, while the other will take on the defensive role of passing the ball to the offensive player. This will be further detailed in later sections.

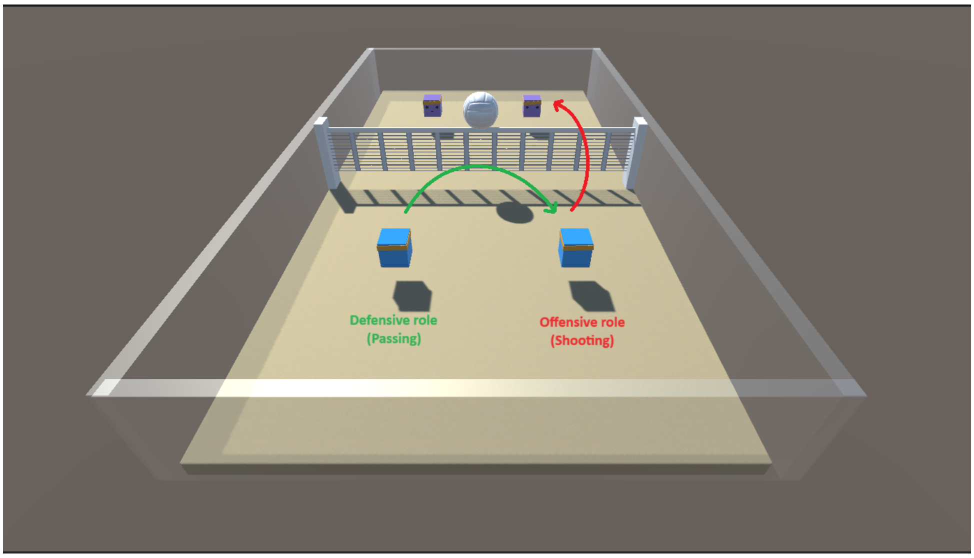


Figure 4.1: General overview of behaviours in the environment

4.1.1 Eliminating disturbance

Before initiating any training, it is crucial to ensure that the environment is sufficiently simplified for the agent to navigate. This involves the removal of unnecessary obstacles, randomness and excessive observations, all of which contribute to the agents learning of its initial moves. In the context of this project, one effective method for assisting the agent is to consistently spawn the ball in the same spot above the agent during each serve, allowing the agent to hit the ball more frequently. This approach aids the agent in scoring more rapidly and accumulating positive experiences, rather than requiring the agent to search throughout the environment to find the spot to hit the ball. Each time the agent successfully hits the ball over the net, it earns a point and associates that experience with a favorable outcome. As the agents become proficient in this skill, random elements in the environment can be gradually reintroduced.

Using the proposed strategy for reward shaping mentioned later in this section, will help reduce the number of rewards early in the training process, which serves as a means of minimizing disturbances. This approach lowers the likelihood of encountering undesired local maxima in the cumulative reward, as many different sources of rewards may lead the agent in the wrong direction, behavioral-wise. Excessive observations can also introduce noise to the agents learning process. Therefore, the choice of observations must be made thoughtfully since the agent relies on the same observations throughout the training procedure.

4.1.2 Proposed reward shaping strategy

To ensure correct behavior, a well-considered reward shaping strategy is required for accurately train a policy to achieve a well-suited behaviour. Starting with an overly complex model can overwhelm agents, especially when they have no prior knowledge of their environment or tasks. It is essential for them to learn the basics before attempting more advanced actions. The proposed strategy for training an agent is based on CL and includes a reward shaping strategy that draws inspiration from [31]. A good approach to start any training session is to define the overall objective from which we can derive key behaviours, in this case:

A 2v2 *volleyball game with two players on each team. The primary goals include positioning themselves correctly, executing successful shots over the net, and making precise passes to a teammate. They should also learn to avoid shooting the ball out of bounds and prevent the opponent from scoring goals.*

Analyzing this statement, the next step is to identify and extract the core behaviours from it:

1. Correct positioning
2. Shooting the ball
3. Pass the ball
4. Avoid shooting out of bounds
5. Avoid opponent scoring

These behaviours serve as contributions to the final behavior. However, implementing all of them simultaneously can be overly complex. So, for beginners, it is crucial to identify and prioritize the most essential skills to learn. Figure 4.2 illustrates a proposed approach for prioritizing these behaviors. The primary goal is to utilize as few behaviors as possible, while still achieving the desired outcomes. The strategy heavily relies on splitting the training in session as in CL, which will be further discussed later in this chapter.

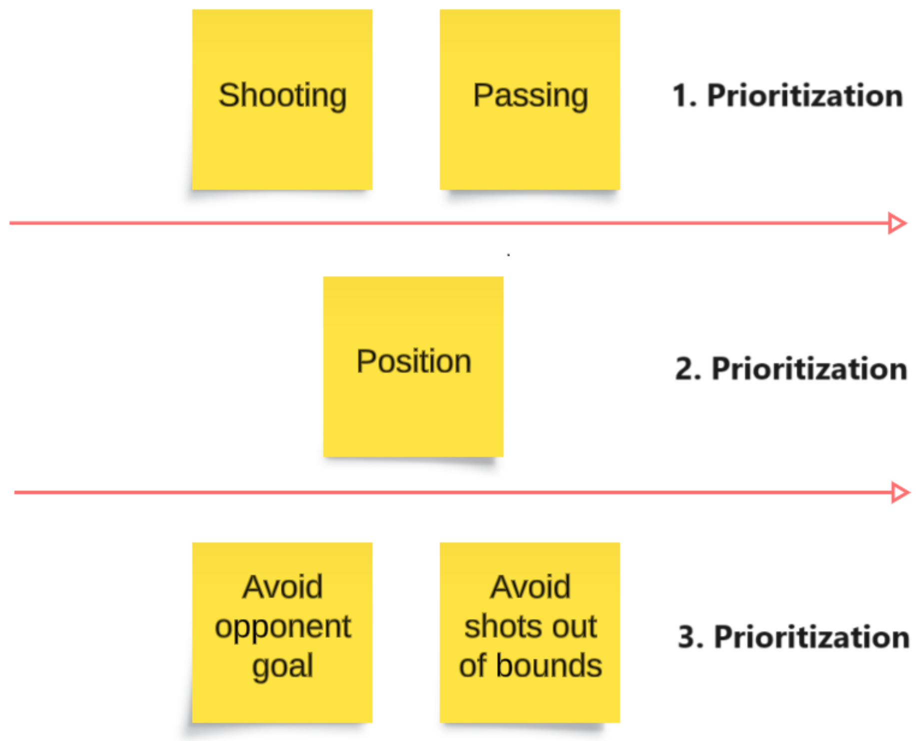


Figure 4.2: Behaviour hierarchy in proposed reward shaping strategy.

The strategy consists of sorting the given behaviours into prioritizing groups and dividing the training into separate sessions, each with a focus on rewarding/penalizing different behaviours. From Figure 4.2, the first prioritizing group represent the agents core behaviour(s). One and only one of these behaviours should be included when initializing the training, and kept throughout the training process. Whenever there are multiple attributes within this group, one can consider training separate models for each core behaviour and coordinating their harmonious place an environment. Introducing multiple first priority behaviors for a single policy is possible, whether it is for one agent or more. If the policy has reached a final state of satisfaction with one core behaviour, only then should a new core behaviour be promoted if necessary. The training should from here be treated as the initial segment of training, meaning all excessive complexity gradually added over time during training should be removed.

The second prioritizing group represents environmental behaviours and promotes the agents movement efficiency in the environment. This often include penalizing of improper positioning, negative behavioral patterns or introducing existential penalties to promote time restrictions. The behaviours belonging to this group are considered important, but not critical, meaning that all behaviours of the second prioritizing group should at one point be a part of a training session. A behaviour of this group should be promoted at an early point of training, but not necessarily from the beginning. Once a behaviour of this group is introduced to the model, it should not be removed.

The third prioritizing group covers strategies to avoid certain unwanted outcomes of the agents final behaviour. These outcomes are cases that can potentially be solved by higher prioritizing groups, but it is worth including in the prioritization model as an overview, and should only be addressed towards the end of training to "polish" the policy. The promotion of these behaviours can be introduced and/or removed, optimally towards the end of the training, but never in the first training session.

This strategy will be used further in the project when implementing the different behaviours.

4.1.3 Selecting observations

As previously emphasized, maintaining simplicity is the key for effective training and optimal performance. As discussed in the micro-course [32] by Joy Zhang, the observations outlined in Section 3.3.3 provide the minimal set of observations necessary for the purpose of shooting the ball over the net. This observation set proved to be effective for the shooting behavior even when transitioning to a $2v2$ game.

However, when it came to the passing behavior, these eleven (11) observations fell short, as they solely described the agents' relative positions to the ball. For the passing behavior, the ability to distinguish positions became crucial, as one agent was required to pass to the right, while the other to the left. This differentiation was not necessary for the shooting behavior, as both agents were encouraged to shoot the ball in the same direction. To instruct an agent about the direction in which it needed to pass, an observation of its positional value was necessary. Another valuable addition was providing an observation of the agent's relative position to a static object in the environment, in which the closest distance to the volleyball net was used. This helped the agent gain a better understanding of its relative position within the environment. For this model the observations will be thirteen (13) components:

- Agent's y-rotation (1)
- Agent's x, y, z-velocity (3)
- Agent's x, y, z-normalized vector to the ball (direction) (3)
- Agent's magnitude distance to ball (1)
- Ball's x, y, z-velocity (3)
- Agent's (int) position (1)
- Agent's magnitude distance to net (1)

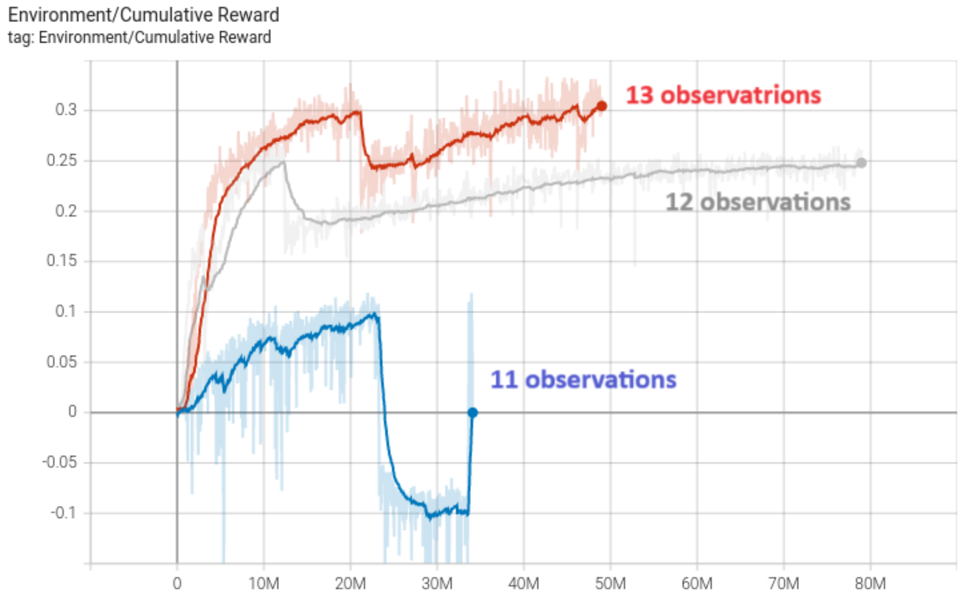


Figure 4.3: Comparison of CR with different observations.

Figure 4.3 illustrates the effectiveness of introducing these components/observations. The blue graphs corresponds with 11 observations, where both agents would continuously try to pass in the same direction. The gray graph lacks an observation reference to a static object and only contains information about the position, while the red graph contains all 13 observations, proving to be the most effective. The sudden change of cumulative reward is due to a transition of one training session to another, and will be further detailed later.

4.1.4 Optimizing training efficiency

Training a proper model is heavily time consuming, thereby optimizing training efficiency is of the essence. While the previous section regards how to avoid starting training in a poorly designed environment, this subsection emphasize how certain modifications and choices can contribute to make the training more effective.

Multiple prefabs

The most well known strategy to speed up training with ML-Agents, is the use of duplicate prefabs [33] in the scene. A more detailed explanation of this can be found at [34]. Figure 4.4 shows the 24 total prefabs used in order to speed up the training. It is important to note that including too many prefabs may cause Unity to operate at a slower pace.

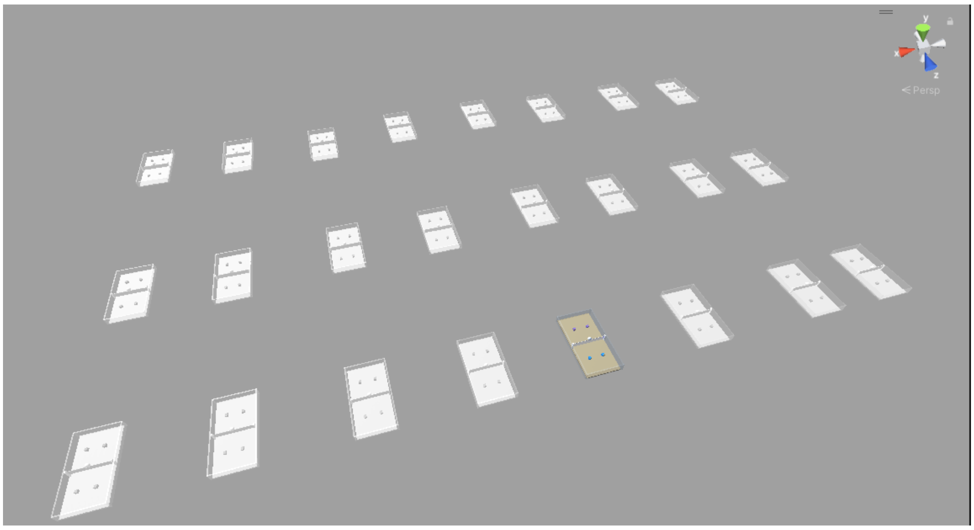


Figure 4.4: Multiple prefabs in the scene.

Hyper parameters

Tuning hyper parameters can be a challenging task, given the considerable time it consumes to thoroughly optimize each parameter for a specific scenario. Unity offers a standardization of parameter ranges along with descriptions [35], which were used as the basis for our hyper parameter choices, as detailed in Section 9.2.1.

The primary parameters that deviated from their default values were the increased *learning rate* (LR) and the adoption of a linear *learning rate schedule* (LRS) instead of a constant one. This decision was made to expedite the agent's learning process in terms of accurately hitting the ball in the right direction. By utilizing a higher LR initially and then gradually reducing it through a linear schedule as the number of time steps increased, the aim was to facilitate faster learning in the early stages for grasping the core behaviors, while allowing for finer adjustments in behavior towards the end.

Neglected observations

From Section 4.1.3, the first thing to notice is that the agents has no information about its teammate nor his opponents. Surprisingly, this approach works well for training a straightforward agent capable of hitting the ball over the net, even when it needs to pass the ball. There are specific reasons for not including observations about opponents. While such observations could encourage a more competitive style of play, the primary goal is to achieve friendly, enduring games where these observations are not strictly necessary.

Regarding the passing behavior, it might seem logical to have observations about the teammate's position. However, with the introduction of various zones, the necessity for these observations diminishes. This will be further detailed in later sections.

The choice of observations depends on the objectives. If you aim to replicate a 'real-world' scenario, these observations may not be relevant. Real-world players are unlikely to have real-time knowledge of details like the distance between them and the ball, making *raycasts* or visual/camera input a more suitable choice. In this specific environment, using *raycasts* is impractical since other agents can easily jump over the rays, leading to confusion, as illustrated in Figure 4.5.

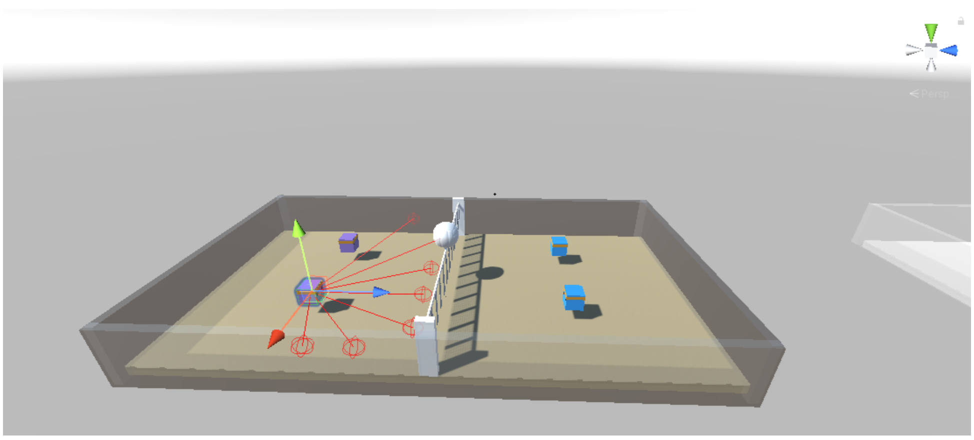


Figure 4.5: Use of raycast observations in the environment.

Splitting into training sessions

By implementing the reward shaping strategy outlined in Section 4.1.2, following a cumulative learning approach has proven to be highly effective. This approach involves breaking down the overall training process into smaller, dedicated training sessions aimed at gradually promoting behaviours to slowly alter the policy. With each training session, complexity is increased through the introduction of new rewards or alterations in the environment. This approach offers advantages such as:

- Minimize the amount of possible local maxima.
- Avoid model over-fitting.

Deciding when to transition to the next training session involves monitoring the game within the Unity editor and observing *Tensorboard* graphs. The transition point is typically identified when the learning curve on the graphs begins to level out, indicating that the current cumulative reward is approaching its maximum potential. In Figure 4.3, you can see a transition from the first training session to the second, marked by a sudden shift in cumulative reward. This happens due to the promotion of a new behaviour, by penalizing the agent for it.

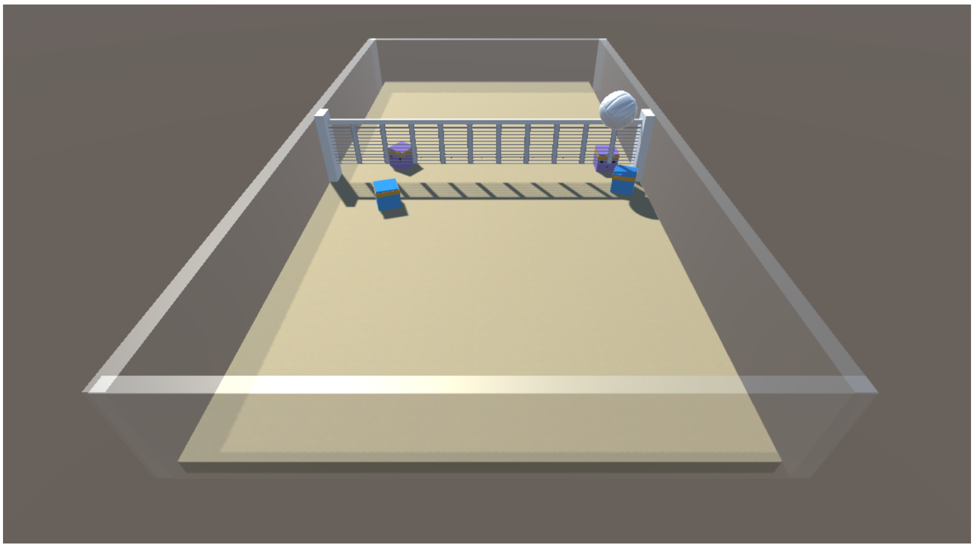


Figure 4.6: Example of over-fitted scenario.

It is also essential to observe how the agents adapt to the reward structure in each training session, as not all behaviors will have a maximum achievable reward. For instance,

Figure 4.6 illustrates the results of extensive training aimed at having the agents consistently shoot the ball over the net. In this case, the agents learned to hit the ball in a way that consistently landed it on the net to continuously score points. As a result, the game consists of the ball getting stuck at the top of the net, while the agents gets confused on what to do next.

4.2 Model training

This section aims to provide practical insights into the implementation of the proposed reward shaping strategy and the design of the environment necessary for training the models effectively. It will primarily focus on the practical aspects of implementing individual behaviors and combining them to create a comprehensive final behavior. The objective is to utilize the strategy outlined in Section 4.1.2 to develop two distinct final behaviors for comparative analysis.

The first final behavior involves the use of two separate models with distinct policies, each dedicated to one of the core behaviors from Figure 4.2, all within the same environment. These models are carefully synchronized to collaborate effectively.

The second final behavior employs a single, more complex model that integrates both core behaviors with the shared policy. This section will delve into the practical steps for achieving these objectives and ultimately comparing the two approaches.

4.2.1 Positional behaviour

Effective positioning is essential, as improper positioning could result in both agents chasing the ball aimlessly, potentially causing disarray instead of promoting teamwork, as demonstrated in the video from Figure 9.1. Given that positioning is a second priority behavior, it will be incorporated into the first training session, while third priority behaviors will be temporarily set aside.

Furthermore, it is important to determine the best methods for executing these behaviors. It is always good to consider multiple ways of execution since, more often than not, the actual outcomes may differ from expectations. Having alternative solutions available is advisable should the initial approach prove ineffective. An example of this could be the following:

1. Correct positioning
    - Punish agent for staying close too each other
    - Create zones in environment and punish faulty positioning

Before making a decision on how to execute the different behaviors, one should take into account the considerations they come with. For positioning, punishing agents for staying close too each other would first of all require more observations which causes more complexity. Inspired by the 2v2 Soccer project, by `deepanshut041` [36], creating the zones seemed like the best option, as it does not need more observations for the agents, and only requires static *GameObject* zones to be added to the environment, thus no added randomness to the model.

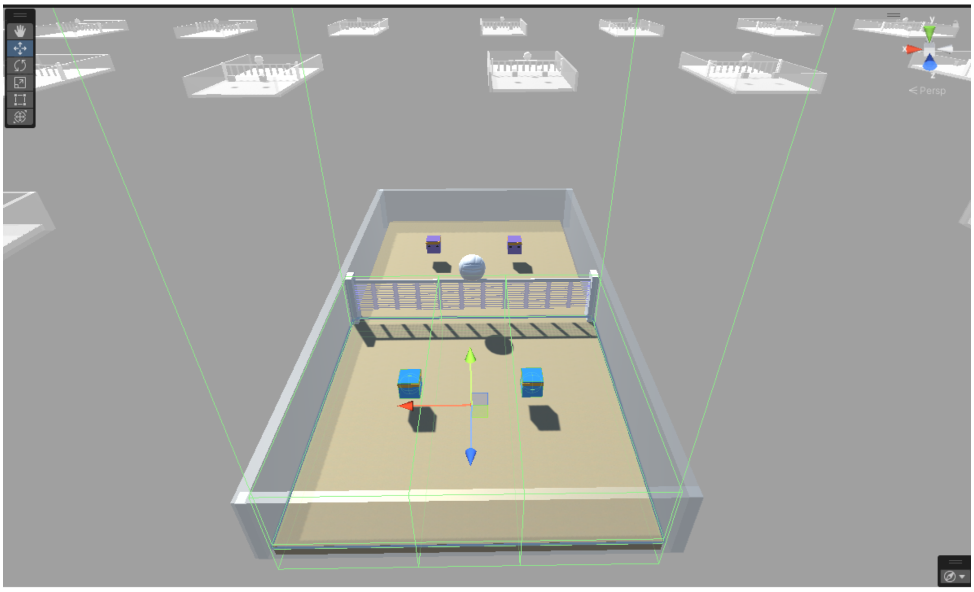


Figure 4.7: Implementation of zones in the environment

Figure 4.7 shows the implementation of the zones. Each court-side contains three positional zones: Left, Middle and Right. Each player will be assigned either Left or Right position at the start of an episode, and the rewards will be assigned as follows:

- Non-correspondent zone (Right or Left)
  - -0.5 upon entry of zone
  - -0.01 every frame agent stays
- Middle or correspondent zone
  - No reward

The Middle zone serves as a free roaming area for both agents in order to create a small overlap where both agents can walk. This avoids implications when the ball lands close to the middle. These rewarding values will be kept throughout the whole training session.

4.2.2 Shooting behaviour

A critical aspect to address is the core shooting behavior, given its status as a first-priority behavior. When it comes to shooting the ball, there are relatively straightforward approaches to shaping rewards. One option is to reward the agent when the ball crosses the net or when the ball lands on the opponent's side of the court. However, there is a challenge when it comes to rewarding the agent for scoring, as it may learn to score most efficiently, resulting in hard shots that the opponent cannot easily counter. This could lead to repetitive and less exciting games dominated by serves and scores. To encourage longer-lasting and more engaging games, we will provide a reward of +1 to the agent for successfully shooting the ball over the net while also penalizing faulty positioning. This approach should be sufficient for reward shaping in the first training session.

As the training progress reach a point where the learning curve starts to flatten or the agents exhibit acceptable behaviors for the intended purpose, we may consider introducing added complexity or addressing specific issues that arise. In our case, the agents demonstrate proficiency in shooting the ball back and forth and maintaining good positioning. However, an issue that affects their performance is excessive jumping, which reduces their mobility. The impact of this issue is evident in the initial segment of the video from Figure 9.2. Typically, smaller issues like this tend to be solved with more extended training, but that may lead to over-training. In our scenario, the plan is to address this jumping issue after the first training session by imposing a small penalty for each jump. The effects of this penalty can be observed in the later part of the video from Figure 9.2.

When a new behavioral trait arises that requires attention, it is a sound strategy to categorize it within the prioritization model as seen in Figure 4.2. In our case, addressing the issue of excessive jumping is accomplished by penalizing the jumping action and not a consequence of an action, which makes it a second-priority behavior. As a rule, once this behavioral penalization is introduced, it should generally be maintained.

In the final part of the video depicted in Figure 9.2, we observe the third training session, which involved penalizing shooting the ball out of bounds while simultaneously removing the penalty associated with jumping. The consequence of this modification is that the agents gradually revert to their undesirable behavior of excessive jumping.

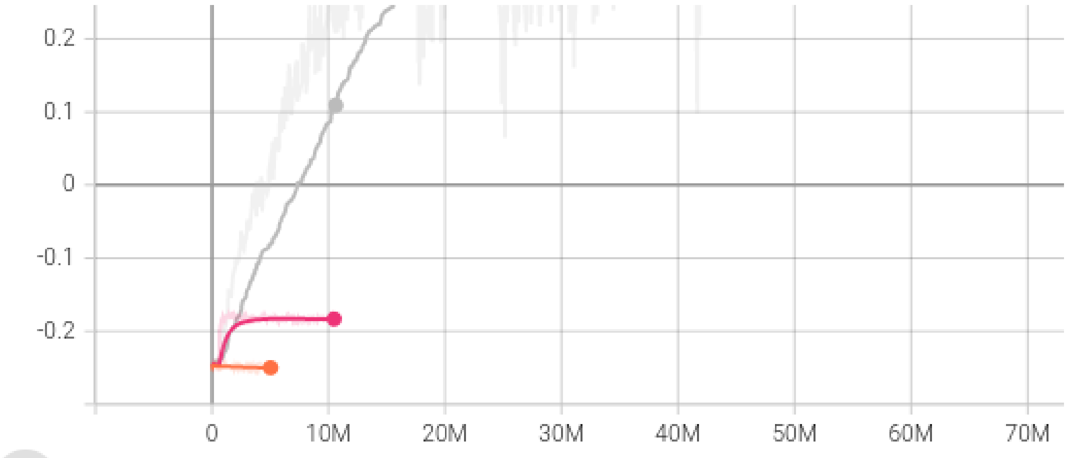


Figure 4.8: Maxima depending on the penalties under consideration.

On the other hand, in Figure 4.8, we can clearly see the significance of introducing behavior rewards too early. Here, there are three different training outcomes, all with shooting as the first-priority behavior and positioning as the second.

The orange line represents the CR of a model with the introduction of a jumping penalty from the beginning. This model never shows progress because the agents quickly learn that jumping is an undesirable action before they realize it can also lead to rewards.

The red line illustrates the learning curve when penalizing shots that go out of bounds. This causes the agents to become overly cautious about when they should hit the ball. The early introduction of these two non-essential behaviors results in an early encounter with a local maximum.

In contrast, the gray line, which includes only the strictly necessary behaviors, demonstrates that the agents gain experience at a more rapid rate. This indicates that early introduction of behaviors prematurely can hinder the learning process and potentially lead to suboptimal outcomes.

To tackle this faulty rewarding, the total training is split into three training sessions. These training sessions include the following reward shaping:

1. Session (0-30M steps)
   - Shooting the ball over the net (+1)
   - Penalize faulty positioning (-0.5, -0.01)
2. Session (30M-90M steps)
   - All previous rewards
   - penalize excessive jumping (-0.1/jump)
3. Session (90M-110M steps)
   - All previous rewards
   - Penalize shooting out of bounds (-1)
4. Session (110M-120M steps)
   - All previous rewards
   - Penalize when opponent score (-1)

4.2.3 Passing behaviour

Due to the passing behaviours status as a 1st prioritization behaviour, a similar strategy to the one used for shooting behaviors should be implemented. However, for agents to pass the ball to each other instead of shooting it over the net, specific environmental changes are required. In order to shape a reward system that encourages passing behavior, agents should be rewarded for directing the ball in a trajectory that is convenient for their teammate to reach. This involves hitting the ball towards the opposite side of their court.

To address this, new aerial zones have been introduced into the environment, as shown in Figure 4.9.

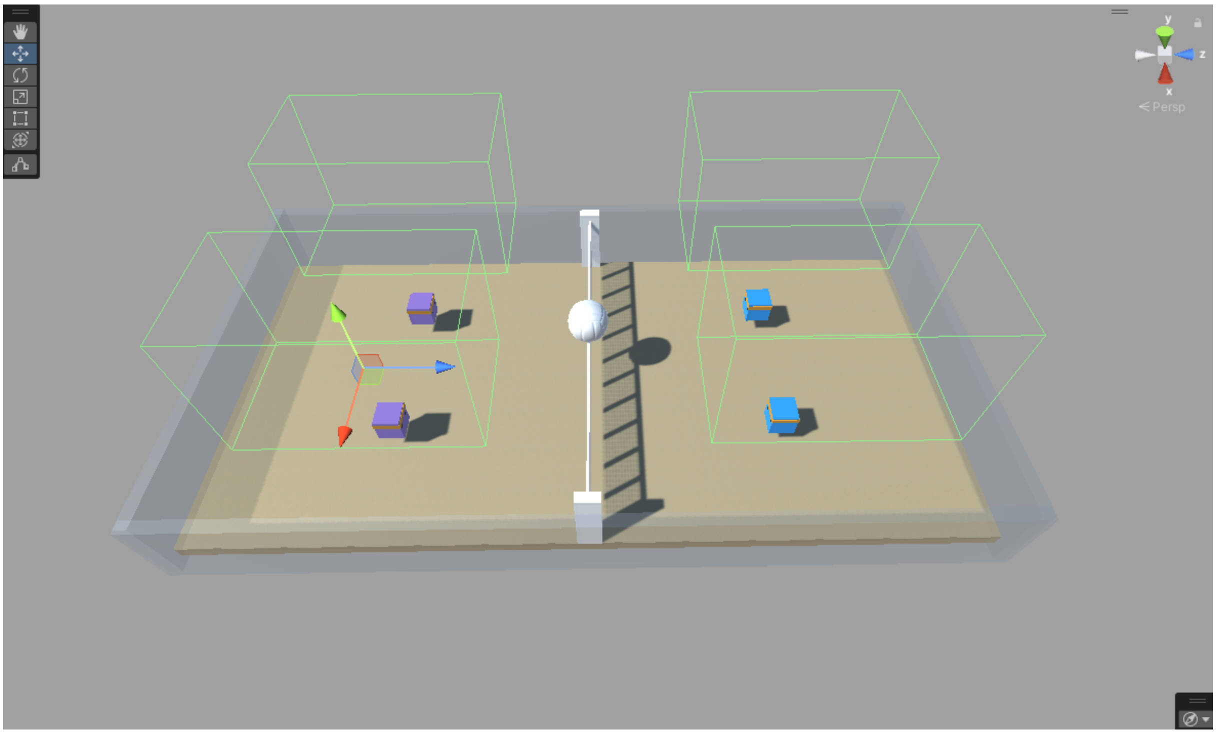


Figure 4.9: *2v2* Environment for passing behaviour with areal zones.

It is essential to note that each agent in the environment is uniquely identified with a team ID and positional ID. This distinction is crucial for training each agent to inherit distinct behaviors within the same model, as discussed in Section 4.1.3. Rewards are structured to favor the agent that most recently made contact with the ball if the ball enters the aerial zone on the opposite side of their own court-side.

Properly scaling the zones is important for achieving desirable results. Overly large zones can lead to imprecise passing, while overly small zones can hinder learning and result in passes that frequently go out of bounds. As illustrated in Figure 4.10, defining the dimensions of the zones correctly creates opportunities for additional rewards by delivering the ball in a trajectory that is more easily reachable for a teammate. This approach leads to more refined passes and facilitates ball reception for the next hitter.

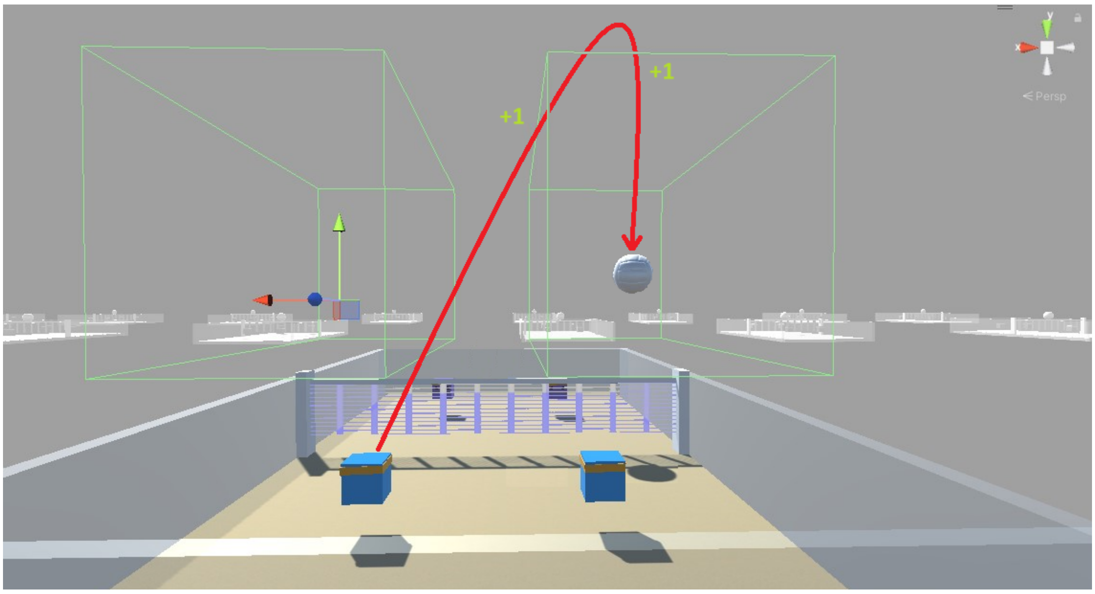


Figure 4.10: Scaling the areal passing zones.

Furthermore, the total training is split into three training sessions. These training sessions include the following reward shaping:

1. Session (0-20M steps)
   - Passing the ball into opposite aerial zone (+1)
   - Penalize faulty positioning (-0.5, -0.01)
2. Session (20M-100M steps)
   - All previous rewards
   - Penalize excessive jumping (-0.1/jump)
3. Session (100M-170M steps)
   - All previous rewards
   - Penalize shooting out of bounds (-1)

If desired, more training sessions can be utilized, for example penalizing the agents if the opponent scores. However, since extensive training is time consuming and the result was sufficient for our purposes, this will be neglected.

4.2.4 Combining core behaviours

In the final stage of the project, we aim to apply the previously described behaviors to the agents, all within the same environment scene. There are two proposed approaches to achieve this:

- **Introduce Separate Trained Models:** In this approach, we introduce the two separate trained models to the environment simultaneously. Each agent per team is assigned its model with a unique policy. One agent is equipped with the trained shooting behavior model, while the other is provided with the trained passing behavior model.
- **Combine Passing and Shooting Behaviors:** Alternatively, we can incorporate both passing and shooting behaviors into the same model by shaping the unique agents' rewards differently. This will lead to a shared policy between the two.

Introducing the two separate models can be easily accomplished by applying the trained shooting behavior model to one agent and the trained passing behavior model to the other. However, incorporating both passing and shooting behaviors into the same model is a more complex endeavour, as it requires the training of a significantly more intricate model. This approach should be divided into multiple training sessions to ensure a controlled outcome.

Using one of the previously trained models as a foundation for incorporating both behaviors, offers time-saving advantages, as we it doesn't rely on training both 1st prioritization behaviors from scratch. Given that the passing behavior took longer to train and the behavior demands more precise ball trajectory, this model is used as the base. This technique is known as Transfer Learning. For simplicity, the total training session of the passing behavior is considered as the first training session, and new reward shaping will be established in the subsequent training sessions:

1. Passing behaviour (0 - 170M steps)
   - Passing the ball into opposite aerial zone (+1)
   - Penalize faulty positioning (-0.5, -0.01)
   - Penalize excessive jumping (-0.1/jump)

- Penalize shooting out of bounds (-1)

2. Session (170M - 200M steps)

   - Reward agent with ID 0 for passing the ball into opposite aerial zone (+1)
   - Penalize faulty positioning (-0.5, -0.01)
   - Reward agent with ID 1 for shooting the ball over the net (+1)
   - Remove Penalty for shooting out of bounds (-1)

3. Session (200M - 230M steps)

   - All previous rewards
   - Reward agent with ID 1 for shooting the ball into the aerial zone in front of it, in the opposite courtside (+1)

4. Session (230M - 270M steps)

   - All previous rewards
   - Penalize shooting out of bounds (-1)

Considering that this model is more complex than the ones from previously, a more intricate application of the proposed reward shaping strategy is necessary. Notably, in training session 2, a third priority behavior is removed. This removal is aimed at minimizing the chances of the agent getting stuck in a local maximum while attempting to learn the fundamental shooting behavior.

In session 3, the shooting agent is additionally rewarded for shooting the ball to the opponent agent with passing behavior. This adjustment is made to prevent a scenario where the two shooting behavior agents exclusively engage in a back-and-forth shooting exchange, promoting a more diverse and dynamic interaction between the agents.

In Figure 4.11, the orange arrows indicate the intended objective. The shooting agents are strategically positioned on opposite sides to facilitate shooting the ball to the agent on the opposing team with passing behavior. With the rewards introduced in session 3, the undesired shots represented by the red arrow in the picture can be minimized.

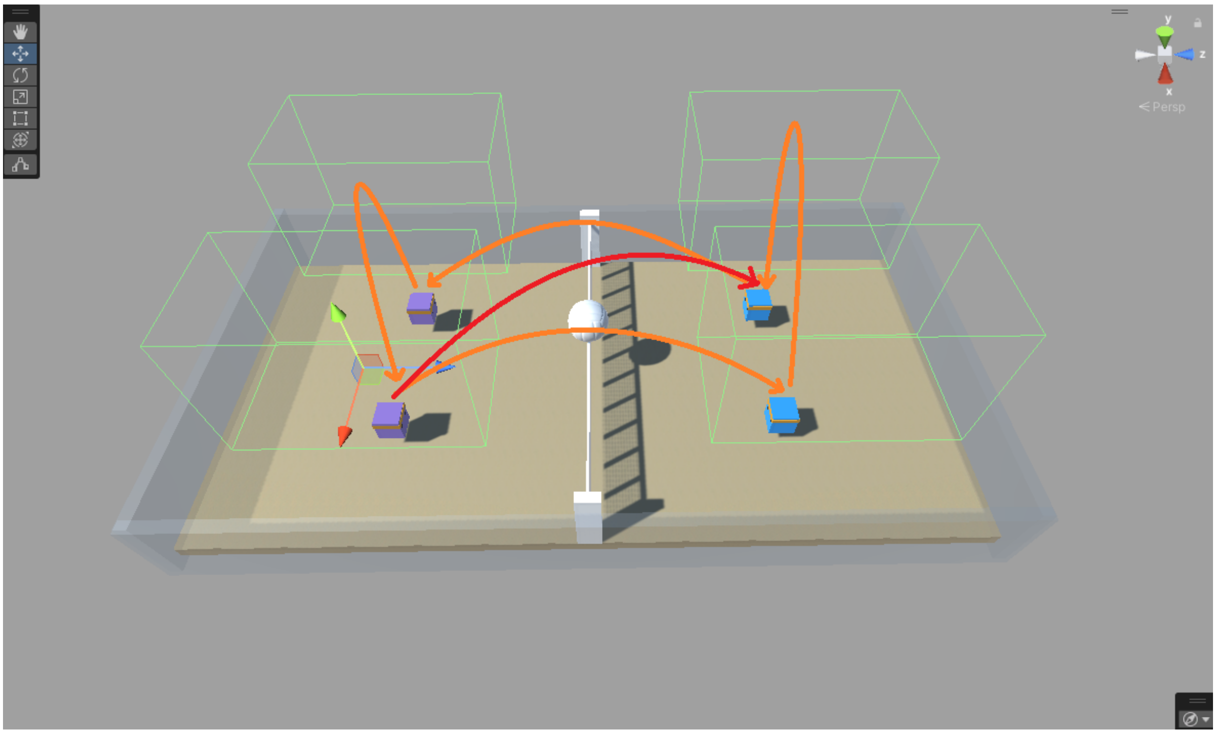


Figure 4.11: Depiction of intended final behaviour in *2v2* Environment

Chapter 5

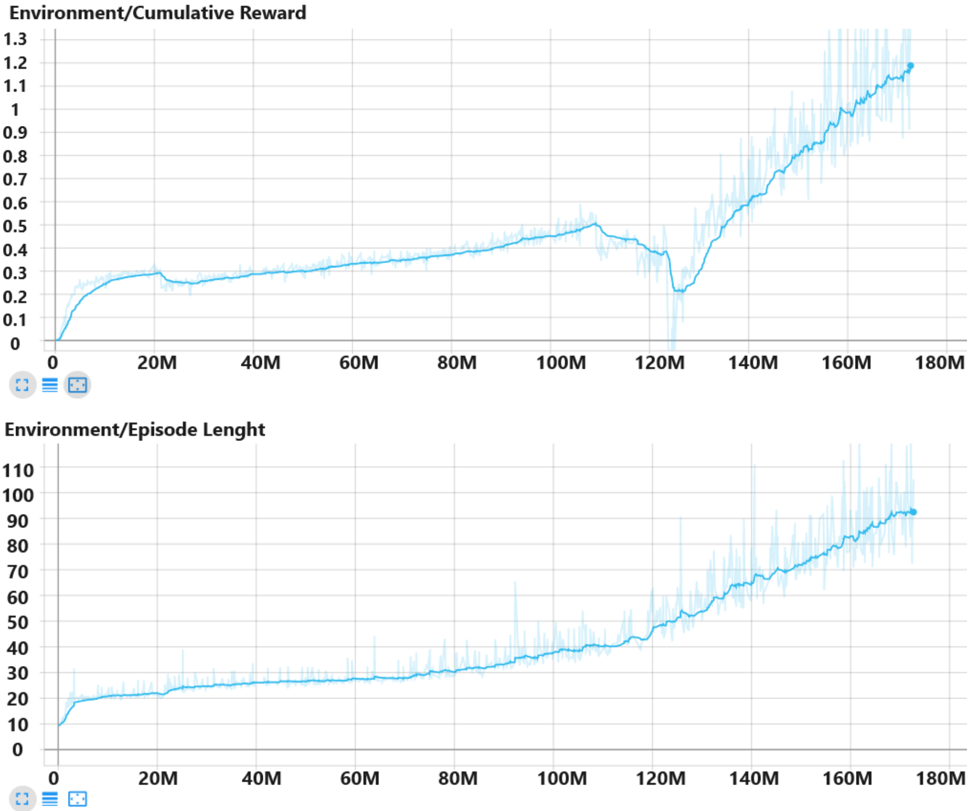
Results

This chapter is intended to present the outcomes resulting from the designs and implementations detailed in the preceding sections. It will showcase the trained models both as individual behaviors and when integrated into the final behavior. Furthermore, this chapter will delve into the interplay between the various training sessions and the cumulative rewards, as well as their influence on the episode duration.

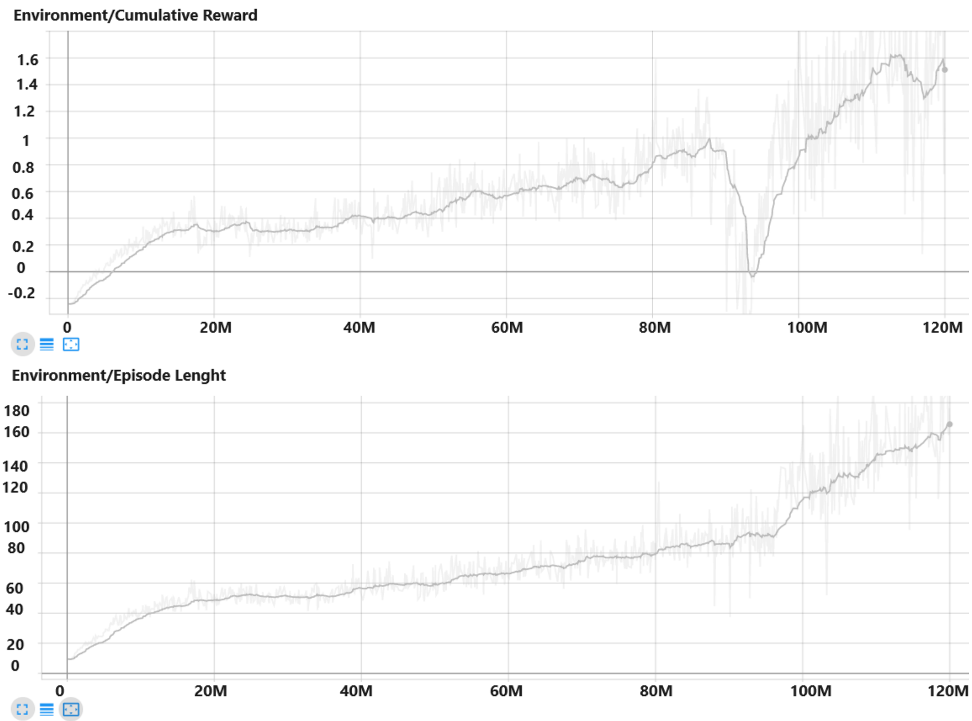
5.1 Evaluating separate models

As we approach the final behavior, we considered two approaches: introducing both behavior models separately in the same environment or creating a single, more complex model. In this section, our focus is on examining the passing and shooting behaviors separately to gain a better understanding of their roles in the final model. In Section 4.2.3, the entire training process was divided into three distinct training sessions, and Figure 5.1a provides a visual representation of these training sessions.

Around the 20 million steps mark, a significant drop in the CR is observed, which corresponds to the introduction of penalties for excessive jumping behavior. A similar trend is noticed around the 110 million steps mark when penalties for shooting out of bounds were introduced. Notably, there are abrupt changes occurring both at 110 million steps and 125 million steps. These changes are a result of increasing the penalty from -0.5 to -1, aimed at making the agents respond to the behaviors more swiftly.



(a) Passing behaviour.



(b) Shooting behaviour.

Figure 5.1: Cumulative reward.

Regarding the shooting behavior, the total training duration is notably shorter than that of the passing behavior, primarily due to the precision required for accurate shots, as depicted in Figure 5.1b. Like the passing behavior, the introduction of jump penalization occurs around the 30 million steps mark. At 90 million steps, penalties for shooting out of bounds are introduced, resulting in a significant drop in the CR. Subsequently, there is a rapid recovery in CR, signifying that the agents are shooting fewer foul balls. This improvement is also reflected in the episode length, which increases notably.

Both of these behaviors are demonstrated in the video associated to Figure 5.2. The initial segment of the video showcases the passing behavior, which displays the successful execution of a suitable ball trajectory during passing. The second part of the video shows the shooting behavior. Here, it is apparent that the agents in possession of the ball exhibit significant activity on the field, while those without possession tend to position themselves away from the action. This reaction is a result of penalties for entering incorrect zones. It is worth noting that this passivity among non-playing agents could pose potential issues, as they have learned to position themselves in a more passive manner when not in possession of the ball.

Figure 5.2: Passing and shooting separate final behaviour.

5.2 Introducing both models to the environment

This Section presents the outcomes when integrating each of the models featured in video from Figure 5.2 into a single scenario. This is achieved by assigning one trained model to each agent on the team, as illustrated in Figure 4.1. Specifically, agents in the right position are equipped with the shooting behavior, while those in the left position are provided with the passing behavior.

Figure 5.3: Final Behaviour, Passing + Shooting separate policies.

Video in Figure 5.3 presents the outcomes of this implementation. A notable observation in this scene is the apparent disconnection among the agents, leading to suboptimal positioning. The shooting agent struggles to receive the ball from the passing agent, and vice versa.

This issue likely stems from the differences between the training environment and the actual scene. The policies the agents have learned are not well-suited to this specific scenario, which requires different decision-making processes due to the variations in ball trajectories. The shooting behavior model had limited exposure to instances where the ball came from the side, while the passing behavior model rarely encountered situations where the ball crossed the net.

As evident in the video, the passing agent positions itself near the court's border, expecting the ball to come from its teammate. This positioning, however, hinders its ability to receive the ball from the opposite court-side. Furthermore, the shooting agent tends to shoot the ball out of bounds more frequently in this scene compared to the one in video related to Figure 5.2, mainly because of the altered ball trajectories, which often originate from the side rather than the opposite court-side.

This approach, characterized by a sudden shift in ball trajectory, is not an optimal solution. The agents' policies are developed through extensive exploration of various instances within their environment, adapting their behavior accordingly. When agents are suddenly placed in a scenario with an abnormal environment or unusual observation values, they struggle to adapt effectively, as their prior experience does not cover such scenarios.

5.2.1 Reducing observations

In an attempt to enhance the final behavior of two models, a deliberate decision was made to exclude the penalty for excessive jumping. The rationale behind this decision was that agents naturally converged towards this behavior, and by disregarding it, there was a potential for an improvement in each agent's individual behavior.

The outcomes of this decision for each individual behaviour can be observed in the separately trained models showcased in videos linked to Figure 9.2 and Figure 9.3. These videos exhibit a behavior similar to the initially trained models but with increased instances of excessive jumping.

Figure 5.4: Final Behaviour without jump penalizing, Passing + Shooting.

Video embedded in Figure 5.4 demonstrates the consequences of eliminating the jumping penalty. It is evident that their cooperative skills have shown a significant improvement. Notably, the agents now tend to stay closer to the boundaries of the court, affording them greater mobility and an improved positional playing style. This contrast can likely be attributed to the presence of multiple policies in the same environment, leading to a higher degree of coordination issues when inaccurately trained, as compared to having only one policy in place.

Due to the excessive jumping still being present, further evaluation of this result will be neglected, despite having a better performance. This result is only showcased as an example of how changing certain rewards during training can affect the final behaviour.

5.3 Shared policy behaviour

In the previous section, each trained model was individually introduced to the same scene, resulting in a slightly disoriented, yet functional behavior. To address the issue of placing an agent in a scene that does not correspond to its policy, an alternative strategy was considered – implementing a single model for both agents. As outlined in Section 4.2.4, the training sessions were divided into four sessions, with the first session continuing from the passing behavior training.

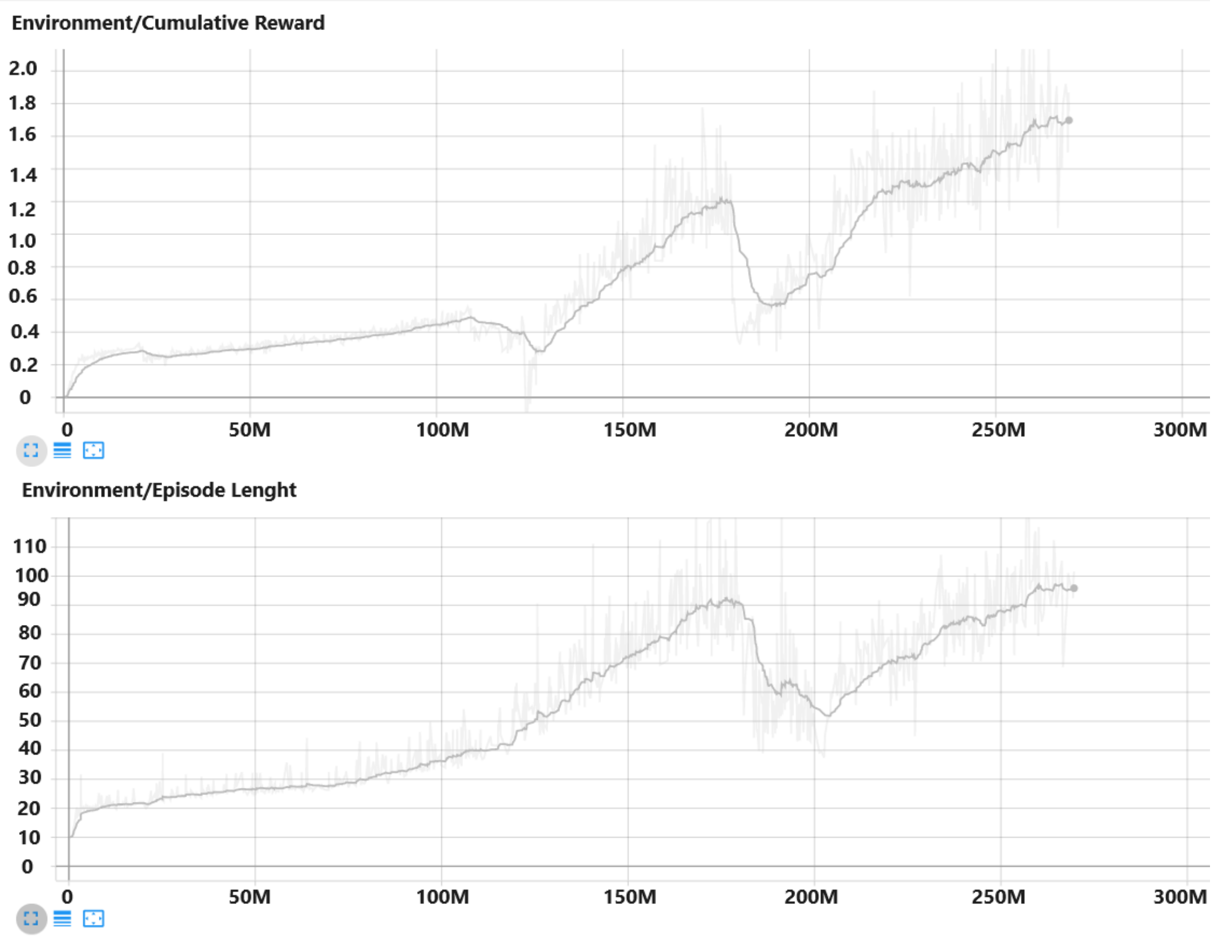


Figure 5.5: Cumulative reward for the shared policy final behaviour.

In Figure 5.5, both the CR and episode length for the training of the combined model are presented. It is worth noting that the episode length remains relatively consistent at approximately 170M step and 270M step mark. This consistency was desired to ensure that the agents play games of similar duration to the other models.

The first notable observation is the significant drop in both episode length and CR during training session 2. This drop was a result of introducing the new first prioritization behavior. However, after only 10M steps the CR started slowly increasing. In contrast, the episode length did not start increasing until the third training session at around 200M steps, where a better ball trajectory was promoted, making it easier for the opposite team to catch the ball. Around 230M steps, the episode length curve begins to flatten out because the agents repeatedly shoot the ball out of bounds. In response to this issue, session 4 was able to further increase both CR and episode length.

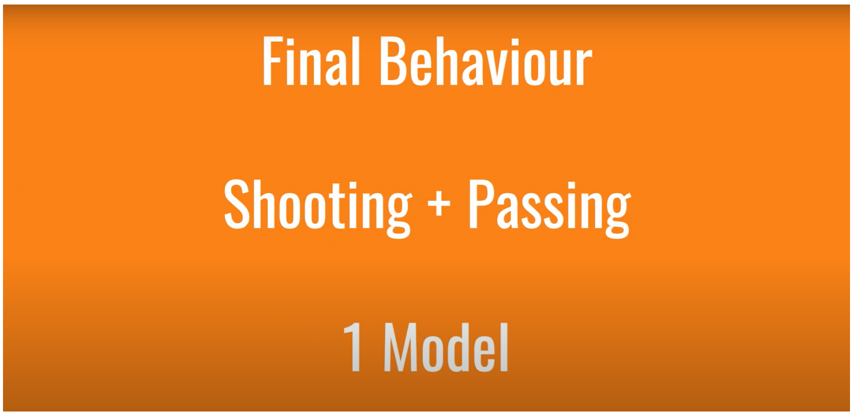


Figure 5.6: Final Behaviour, Passing + Shooting shared policy.

Video linked to Figure 5.6 showcases the final result of the trained model with two distinct first prioritization behaviors. This result demonstrates significantly more cooperative tendencies among the agents compared to when two separate models were introduced. Both agents exhibit effective coordination, positioning themselves appropriately depending on the ball's trajectory.

One notable aspect of their behavior is that they still tend to position themselves along the borders of the court. This behavior is likely a remaining trait from the first training session. However, due to the improved ball trajectories, the agents are able to handle catching the ball effectively.

5.3.1 Comparing results

Given that the proposed reward shaping strategy introduced multiple training sessions, dependant on the user to interfere and take decisions on how and when to transition from one training session to another, the results may differ to a high degree both depending on how good the agents are performing, and depending on how the user decides to interfere. Because of this, there will always be potential to improve, by giving the user freedom to choose when and how to interfere with each training session.

One approach could involve shortening each training session or reducing the incorporation of less prioritized behaviors. Although these adjustments might diminish the overall training duration, they could potentially impact performance adversely. As previously mentioned, evaluating the agents' performance involves monitoring progress through *TensorFlow* charts and observing the evolution of playstyle in Unity. However, this can also be misleading.

Comparing the two models with separate policies, with the shared policy model based on tensor graphs would be inaccurate, as the separate policy agents are operating in an unfamiliar environment, performing worse then what their respective CR and episode length suggest. Thus, a more accurate basis for comparison lies in observing the agents' actual gameplay.

Table 5.1 presents data collected from observing the agents' performance in each trained model. Evaluation criteria align with the objectives outlined in Section 1.2. The data reflects the overall performance of all agents on the court, including both teams. Through 100 volleyball games, the following events were counted to get an overview about each models performance:

- **Shots:** how many times the ball was successfully shot over the net per cent games,
- **Passes:** amount of successful passes per cent games,
- **Bad position:** amount of games an agent entered a faulty zone
- **Shots OOB:** amount of games ending with shots *out of bound* (OOB).

Objective	Shared policy	Passing	Shooting	Separate policy
Shots	331	15	934	162
Passes	298	682	8	74
Bad position	11	0	35	55
Shots OOB	26	87	68	12
Training time	270M	170M	120M	-
CR	1.7	1.2	1.5	-
Average Ep. length	95	95	165	-

Table 5.1: Policy performance comparison.

Keep in mind that whenever there is a shot OOB, it is not necessarily negative as an episode can end by either the ball hitting the ground, going OOB or the episode reaching the maximum 2000 steps. As none of the episodes reached the maximum step count, those that didn't conclude with the ball going OOB inevitably ended with the undesired outcome of the ball hitting the ground.

5.4 Discussion

This section delves into the discussion surrounding the completed project, aiming to highlight different aspects of the project and evaluate the pros and cons of its implementation. At every stage of the journey towards achieving the final product, it is essential to assess how to create the desired outcome most efficiently and effectively.

Identifying the best approaches during the project's execution is not always straightforward, as better choices and strategies may become apparent in hindsight. In this discussion, key factors to consider will include time constraints, resource availability, and overall efficiency. Consequently, this chapter is dedicated to several key objectives: justifying the decisions made to achieve the best results, exploring alternative methods that were considered, and offering suggestions for improvements.

The following presents the assessment of advantages and drawbacks about the various aspects of the completed tasks in the project:

1. **Unity Game Engine:** Execution and positive features

    - **User friendly**. Unity provides a user-friendly interface for designing and customizing virtual environments, which made it easy and fun further developing.
    - **Graphics and Physics**. Unity's advanced graphics and physics engines enabled the creation of visually engaging and physically accurate environments, which was crucial for training agents in complex scenarios
    - **Convenient scripting**. Unitys' extensive built-in components, along with Visual Studio and C#, made it easy to create new scripting for game objects in the scene.
    - **Record local, expert demonstrations**. The Unity Editor includes a Play mode which enables a developer to begin a simulation and control one or more of the agents in the Scene via a keyboard or game controller. This enabled for easy testing before starting a training session.
    - **Record large-scale demonstration**s. One the most powerful features of the Unity Editor is the ability to build a Scene to run multiple platforms. This drastically speeds up training.
    - **Free cost**. Unity is free for all developers, as long as your income is below a set limit. As a poor student this comes in handy.
    - **Community**. Unity also has a large and active development community so that creating new and diverse environments is easy with an expansive array of off-the-shelf assets.
    - **Importing Projects**. This thesis used an already made project as reference for further development. Importing this project was easily done with Unitys' asset ad-on compatibility.

    Proposals for improvement and less positive features for Unity game engine.

    - **New pricing plan**. Unity introduced September 12, 2023, a new pricing plan about the changes to the Runtime Fee policy for 2024. How this will affect different developers in the Unity community varies. However, it can definitely economically affect ML developers that develop on a large scale.

2. **ML-agents toolkit:** Execution and positive features

    - **Open source**. Ability to create a simulated environments using the Unity Editor and interact with them via a Python API. This contributed to the user-friendliness and accessibility.

- **SDK**. The toolkit provides the ML-agents SDK which contains all functionality necessary to define environments within the Unity Editor along with the core C# scripts to build a learning pipeline.
- **Easy to learn**. It includes a set of example environments, which made it quick and easy to understand how to use the toolkit in practise. Reviewing these example beforehand made it much easier to start working on the project.
- **RL algorithms**. The toolkit provides state of the art RL algorithms that can easily be used, including PPO, SAC and more. In this thesis PPO was used, but for future reference, it could be interesting to explore alternative ones.
- **Policy flexibility**. Any number of agents can have a policy with the same behavior name. Additionally, there can be any number of behavior names for policies within a scene, which enabled simple construction of multi-agent scenarios with groups or individual agents executing many different behavior types. This allowed for the testing of two different approaches to the final behaviour.
- **Rewards**. The reward function, used to provide a learning signal to the agent, can be defined or modified at any time during the simulation using the Unity scripting system. This made it easy to divide the training into sessions.
- **Graph visualization**. The Python API provides a Tensor Graph visualization tool that makes it easy to see progress in the current scene, as well as compare model results.

Proposals for improvement and less positive features ML-Toolkit.

- **Simulation Speed**. It is possible to increase Unity ML-agents simulations up to one hundred times real-time. However, this was not done in this project due to computational resources availability. For future reference, this could potentially speed up training, but requires a powerful computer.
- **Algorithm research**. As mentioned, for this thesis PPO was chose as this performed superbly in similar competitive ball games such as $2v2$ soccer as described in [3]. However, in order to explore further, an idea is to try other algorithms such as SAC, as this shows promising results.

3. **Environment design:** Execution and positive features

   - **Implementing zones**. Creating zones and collision trigger points to apply a reward to an agent worked great for the purposes of creating desired ball trajectories and creating positional behaviours.

- **Middle/Neutral zone**. Having a neutral middle zone in the environment proved to be much more effective than just having two zones. This eliminated the agents fear of approaching the middle of the court, which initially was a problem whenever the ball landed in that area.
- **Enter and stay penalty in zones**. Having two different penalties on the zones, one higher for entering the wrong zone and a small existential penalty for every frame the agent stayed in the wrong zone, was an effective strategy to encourage the agents to position themselves, but be able to actually enter the wrong zone if it was only for a brief moment.
- **Simplistic observations**. The observations given to the agents was minimal, as introducing too many observations can serve as a disturbance for the agent. This lead to a faster learning among the agents'.
- **Eliminating disturbances**. Elimination of disturbances early on in the training is a known trick in RL. By reducing randomness of ball spawn location significantly sped up initial learning, making the agents quickly master hitting the ball at the serve.
- **Structured reward shaping**. Having a structured reward shaping strategy as the one proposed, helps keeping track of and structuring the complicated process of shaping a complex policy. The objective with this proposal was not to make a method that needs to be strictly followed to every detail, but rather simplifying the process, making it easier to understand the agents adaptability and learning.

Proposals for improvement and less positive features for environment design.

- **Alternative observations**. There was no observations about the current position of any agent. This could potentially lead to a better result over time, but may reduce the agents' rate of learning.
- **Reward shaping strategy can be misleading**. As mentioned, the purpose of the proposed strategy was not to define a strictly necessary method to follow. In reality, the strategy is heavily dependent on the desired objective and environment. The same strategy may not work in different circumstances. An example of this was dealing with the excessive jumping behavior in two different models, compared to one. When an undesired behaviour pattern emerges during training, penalizing it could potentially result in disruptions the agents overall performance.

- **Lack of variety in observation types**. There are three types of observations availible; vectors, raycast and camera input. Raycasts was neglected due to impracticality. However, it would be interesting to use camera input, as using frames as observations could lead to interesting and potentially better results. This has previously proved to be effective in third person racing games [37], where the camera is locked on the player.
- **Lack of observations in shooting model**. As having 13 observations worked well for the passing behaviour and shared policy, it would be interesting to see how the shooting behaviour would perform with the two additional observations.

4. **Model training:** Execution and positive features

- **Splitting training sessions**. Instead of having one long training session, which is often impractical when creating complex policies. Using CL to split the training into smaller sessions enabled more controlled shaping of the policy.
- **Planning training sequence**. As described in the reward shaping strategy, creating a statement of the objective and extracting core behaviours from it helps simplify the problem and plan the sequence of training sessions. This turned out to be quite helpful as training should not be conducted without proper revision in order to not waste time training with bad reward shaping.
- **Surveilling training progress**. When splitting into smaller training sessions, its important to observe both tensor graphs and performance in unity editor. This helped choosing the best time to transition into the next training session.
- **Different solutions for behaviour**. Having multiple options for addressing a certain behaviour is a good idea, as the first solution often times don't work as expected. This allows for faster trial and testing, without having to go back to the drawing board for every failure.
- **Identifying separate agents**. Introducing unique agent ID and applying rewards individually enabled creating multiple behaviours with a single policy.

Proposals for improvement and less positive features for model training.

- **Long training time**. The total training time for the final model required quite a long time to assess, even with multiple prefabs. The final model reached 270M steps, and still had potential for improvement. When the desired behaviour is complex, it is necessary to split the total training up into smaller training session for better results, but at the cost of time. Its also worth noting that whenever more agents are added to the environment, the training time for the same outcome can more then double. In the original 1v1 volleyball project, a consistent shooting behavior was achieved within 20 million steps. However, upon adding two more agents to emulate the shooting behavior illustrated in Video linked to Figure 9.1, the training requirement escalated to 40 million steps. Furthermore, introducing positional behavior necessitated over 80 million steps to attain similar consistency in the overall playstyle. These instances underline the escalating training duration corresponding to the complexity and expanded scope of desired behaviors within the system.

- **Human-in-the-loop Training**. Imitation learning plays a significant role in mainstream field of research, and with manual play it is possible to speed up training. As controlling the player manually proved to be quite difficult, it was decided to avoid this. However, for games that require less precision to execute, it can be an advantage.
- **Excessive jumping**. In every training session, a common behaviour among the agents was the excessive jumping. Much effort was put in to try to prevent this, without excessive complexity. Dealing with the behaviour, especially left its mark on the scene with separate policies. Ignoring this behaviour patters proved better results for these models in the same environment.

5. **Hyper parameters:** Execution and positive features

- **Ease of tuning PPO**. For this thesis the PPO algorithm was used. This algorithm proved easy to implement as it contains fewer hyper parameters to tune, and on average performs good and more consistently then other traditional algorithms.
- **Standard template for parameters**. Unity provides a standard of ranges for the parameters which was used. These worked well for our purposes.
- **Linear vs Constant LRS**. Having a linear LRS instead of constant, while initially having a higher learning rate then recommended, helped speed up the learning curve in the initial training sessions, where the core behaviours are promoted.

Proposals for improvement and less positive features for hyper parameters.

- **Time Consumption**. Even though PPO include less parameters to be tuned, tuning each individual parameter was still a time consuming task, given that fully training a model could take days to complete. For future reference, it would be interesting to see which parameters could be tuned to achieve better performance and faster results.

6. **Results:** Execution and positive features

- **Comparing different approaches**. Having two different approaches showcases the different results, as well as emphasising that there are multiple ways of achieving the same objective. The model with a shared policy was more visually pleasing as the cooperation between the agents was much more harmonic. What made a difference between the two was the increased complexity with having two policies in the same environment. Simplicity and structure is the key to shaping desired behaviour.
- **Achieving better results removing rewards**. Removing the jumping penalty when introducing the separate models into the same environment, lead to the discovery of an improved behaviour. This emphasizes the importance of not introducing unnecessary rewards into a complex model.

Proposals for improvement and less positive features for the result.

- **Performance of separate models**. The performance in each passing and shooting models was quite good when put in their respective separate environment. However, when being put into the same environment both agents performed significantly worse. Their respective policy did not respond as good to an unfamiliar environment. A way to deal with this would be to introduce more instances where the agents need to respond to similar instances, such as making the ball spawn with a passing trajectory from the side when serving, simulating a pass. This would help the policy for the shooting behaviour adapt to more such circumstances.
- **Lack of robustness**. After training the final behaviour, not much variation was added to the environment. Towards the end of training a model, it is generally a good idea to add random elements to the environment to increase the robustness of the policy. Examples of this could be wider range of spawn points for the ball or agents.

Comparison of results: From Table 5.1:

- **Shared policy.** The model with a shared policy shows a good ratio between passing (298) and shooting (331), meaning that on an average game, approximately 3 successful passes and shots were made, with a limited amount of faulty shots as outlined in Figure 4.11. 11 out of the 100 games included at least one agent entering a wrong zone, which is considered a good ratio.

- **Passing**. Out of the 100 games, 682 sucessful passes was made, meaning the average game included almost 7 passes. There where 0 instances of faulty positioning among the agents, which reflect their play style, as seen in Video embedded in Figure 9.4 with the agents positioning themselves near the outer rims of the court. The high amount of shots OOB (87) makes sense as there are less margin for error when passing, as the court is not very wide, and the agents are encouraged to pass the ball with a high trajectory.
- **Shooting**. Out of the 100 games, 934 successful shots was conducted, meaning the average game included between 9 and 10 shots back and fourth. The amount of bad positioning are significantly higher in this model (35). This makes sense, as in this case the playing was usually between two of the agents, while the other two agents where patiently waiting for the ball to enter their side of the court, leading to the agents outside of the action learning to push their boundary in order to be able to play. This resulted in more frequently entering the wrong zone. Another noteworthy observation is the low amount of shots OOB (68), compared to the passing behaviour (87). This can be considered negative, as it means that the policy is less robust when catching incoming shots from the opponent.
- **Combined.** When introducing two separate policies to the same environment an interesting result emerges. Firstly, the ratio between successful passes (74) and shots (162) indicates the agent with shooting behaviour has more dominant playstyle. The amount of faulty positioning instances also reflects this, as all of them was committed by the shooting agent. When a policy has more aggressive decision making tendencies then the other, it will lead to less harmony among the agents' play style and less cooperation, with each agent focusing on their immediate task instead of playing for consistency. The low amount of shots OOB (12), indicates a severe lack of robustness to receiving shots from the opponent or passes as well, as 88 of the 100 games ended in the ball hitting the floor.

Chapter 6

Impact

AI, ML and RL are already having an impact on many areas of society, and further advances in AI research are likely to precipitate much greater impacts. AI is making its debut on more and more domains on a rapid pace, due to its multi-functionality potential. In this chapter, the project impact is presented, by highlighting its possible impact on a social level, especially regarding education, healthcare, entertainment and development.

6.1 Social impact

Reinforcement learning holds the potential to bring significant social impact across various domains. This section aims at shedding light on the social impact potential of this thesis.

ML-Agents reinforcement learning frameworks in Unity offers an interactive and dynamic environment for educational purposes. Students and developers can utilize RL to grasp complex concepts, experiment with algorithms, and gain hands-on experience in a simulated setting. This rise of RL education can contribute to a more widespread understanding of AI principles. The educational benefits this brings are significant, and helps creating engagement among students, independent developers and bigger companies

RL models developed within 3D development engines such as Unity, have the potential to revolutionize medical training and healthcare simulations. These models can be employed to create realistic scenarios for training medical professionals, refining surgical techniques, and developing AI-assisted diagnostic tools. The impact extends beyond training, as RL can aid in optimizing treatment plans and resource allocation within healthcare systems. In [38], A variety of applications of RL approaches in the healthcare field are documented, including Precision Medicine, Dynamic Treatment Regime, Personalized Rehabilitation, Medical Imaging and more. Creating a game-like experience such as a volleyball game does not directly impact the field of medicine, but innovation and promotion of the application can make an impact and spread awareness about its usefulness. This project is aimed to the contribution of exactly this. With a proposed strategy to structure the shaping of rewards over the training sessions, can help new developers get a better grasp on the shaping of a policy.

Unity's RL capabilities play a pivotal role in shaping the gaming and entertainment industry. Intelligent and adaptive NPCs can enhance gaming experiences by providing more realistic and challenging interactions. This not only elevates entertainment value but also contributes to engagement within the field and the advancement of AI-driven narrative experiences within virtual worlds. One of the more well known examples of this is OpenAI Five, which became the first RL system to defeat the world champions at an e-sports game [39]. OpenAI Five demonstrates that self-play reinforcement learning can achieve superhuman performance on a difficult task. Given these advancements, maybe one day AI robots will be able to beat the world champion in volleyball as well.

RL in Unity is instrumental in the development and testing of autonomous systems, such as self-driving cars and drones. The ability to simulate complex environments allows for extensive training scenarios, enabling these systems to learn and adapt without posing real-world risks. This has profound implications for the future of transportation, logistics, and urban planning. As this thesis sheds light on the capabilities of RL within a simulation environment, hopefully it will catch the attention of great developers to inspire more use of RL in other fields such as autonomous systems.

6.2 Budget

The current project is a research project, so it only includes the software used and the hours dedicated to develop the project. As the master final thesis has 12 credits, the total dedicated hours by the student are 360 hours.

6.2.1 Personal costs

This personal budget has been based on the gratification of an internship. The UPC has set a minimum amount 10 €/hour for master's degree students. That is a total of 3600 € over a time period of 6 months, with a working amount of approximately 14 h/week over 26 weeks.

Concept	**Hours**	**Cost in €**
Research	50	500 €
Coding and developing	90	900 €
3D design	25	250 €
Testing and troubleshooting	70	700 €
Training	100	1000 €
Documentation	25	250 €
Total	360	3600 €

6.2.2 Material costs

To develop this project a computer is required. Due to high computational usage while training, a computer with a availability for good air cooling and sufficient processor is required for optimal performance. The computer used in this project had a *Intel Core i7 processor*, which worked well. Also an extra monitor was bought to be able to work on one screen, while training on the other.

Material	Cost
Computer	800 €
Monitor	100 €
Total	900 €

6.2.3 License costs

In order to develop this project the vast majority of software are free. However, there are some guidelines to the license of Unity, stating that if you have 100.000$ or higher yearly salary, Unity License is no longer free of use.

Software	Personal use cost	Commercial use cost
Unity License	0 €	0 €

6.2.4 Energy costs

A computer uses on average between 50W and 200W, so 100W would be a reasonable assumption, considering the large computational efforts while training in Unity. The average price for electricity for 2022 in Spain was 0.2966 €/KWh. From this we can derive an estimated energy cost.

Device	Consumption
Computer	0.1KW
Computer Time	360h
Energy cost	0.2966 €/KWh
Total Cost 360h	10.7 €

6.2.5 Total cost

Personal costs	3600 €
Material costs	900 €
License costs	0 €
Energy costs	10.7 €
Total	**4 510.7 €**

6.3 Environmental impact

Even-though this project is based in a software that by itself will not have an environmental impact, we cannot omit that the use of it implies the need of a computer to execute it. For this reason, the use of this software will have an environmental effect caused by the computer.

According to the *International Energy Agency* [40], it is estimated that the emissions of 2022 was 241 g CO_2/KWh, so with 0.1KW, the use of the application will cause 24,1 g CO_2/h and a total emission of 8676g of CO_2.

Chapter 7

Conclusions

In conclusion, this master thesis is focused on creating a structured strategy within RL, in order to train AI agents to play volleyball within a virtual environment using Unity Game Engine and the ML-agents toolkit. The steps towards a desirable outcome and good shaping of policy, included considerations such as environment design, model training, reward shaping, and the evaluation of results. The objective was to create agents capable of acting and playing in a cooperative manner, including behaviours such as passing, shooting and correct positioning, ultimately aimed at achieving long-lasting and consistent gameplay.

Throughout this research, the implementation and evaluation of the different approaches, including the creation of multiple and shared policy, shed light on some of the possibilities available when using ML-agents. It shows how the ad-on package allows for flexibility when designing distinct behaviours within the same environment, while also shedding light on the pros and cons of each approach. The breakdown of training sessions, reward strategy, and the adjustments made during training provides a more structured and more easily understandable approach of shaping desired behaviors within a complex environment.

The Unity Game Engine was a great tool for providing a user-friendly interface and powerful tools for environment design, leveraging advanced graphics and a realistic physics engine. Meanwhile, the ML-agents toolkit provided a good user experience to quickly get familiar with RL, by providing easy access to model training, enabling the implementation of RL algorithms and policies essential for teaching intelligent and robust decision-making to the agents.

Noteworthy observations emerged during the training process, emphasizing the importance of having a structured reward shaping strategy. By gradually adding complexity for each training session, less possible negative behavioural patterns could emerge, effectively limiting the margin for the behaviour to converge towards a undesired local maxima in the CR. The comparison between models with separate policies and the one with a shared policy shows the challenges associated with introducing a policy to unfamiliar scenarios, highlighting the need for comprehensive training to increase robustness when dealing with multiple policies that needs to interact. In this sense, the shared policy model outperformed the the separate policies due to being better adapted to the cooperative aspect of the game.

Despite the achievements in cooperative behaviors between agents, this research also identified areas for improvement. Challenges such as naturally emerging undesired behaviours, limitations in adapting a policy to unfamiliar environments, and the need for increased robustness in the policy decision-making process by slowly increasing complexity, rather then starting training in a complex environment.

Finally, this thesis contributes valuable insights to the educational aspects of training AI agents for complex tasks such as cooperative gameplay in dynamic environments. This research not only provides a glimpse into the process of training AI agents in a virtual environment, but also creates a foundation for other developers interested in utilizing ML for further explorations and refinements in the engineering field.

Chapter 8

Future Work

Throughout this master thesis, the milestones outlined in Section 1.2 have been thoroughly addressed and successfully achieved. This resulted in the development of a game environment where artificial agents actively participate, with the added feature of allowing manual control of one player, providing the potential for an interactive gaming experience. However, the project has much potential to improve, some of which have been identified in Section 5.4. This section aims to delve deeper into the opportunities for advancements and future work for this project.

Primarily, the proposed reward shaping strategy stands as one of the project's more intriguing aspects. It is observed that developers often resort to a trial-and-error approach when structuring rewards for agents. Unfortunately, comprehensive documentation on reward strategies concerning complex behavioral agents, especially in the realm of cooperative behavior, remains limited. it would be interesting to see the adaptability and robustness of this strategy in other projects, exploring its effectiveness in various scenarios.

On the other hand, if the primary objective of this master thesis was to create an engaging video game that attracts players, the project may fall short. The focus has been primarily on the functionalities related to ML-agents, with minimal attention devoted to aesthetics, graphics, and overall game-play experience. A logical progression for this project would involve further developing the game-play experience by enhancing aspects such as aesthetic level design, enhanced graphics, smoother character control, exciting point system, diverse difficulty levels, etc. Expanding the project beyond its educational scope to create a refined game design could lift it from a beta-level prototype to an engaging, audience-attracting project.

Refining the human participation aspect of the game, is the key to enhance the total experience. Implementing user-friendly controls that enables easier mastery of the game controls by human players, could serve as an effective method to enhance agent training. As human players engage, the AI opponent agents would dynamically adapt and improve by learning from the human player's actions and errors. This concept introduces an adaptive learning mechanism for agents, a feature rarely explored in conventional video games, where AI adapts to individual player play-styles.

Chapter 9

Video results and resources

9.1 Videos

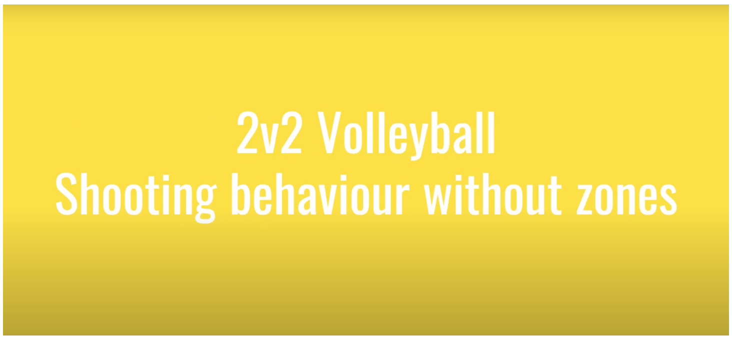


Figure 9.1: Shooting behaviour w/o zones

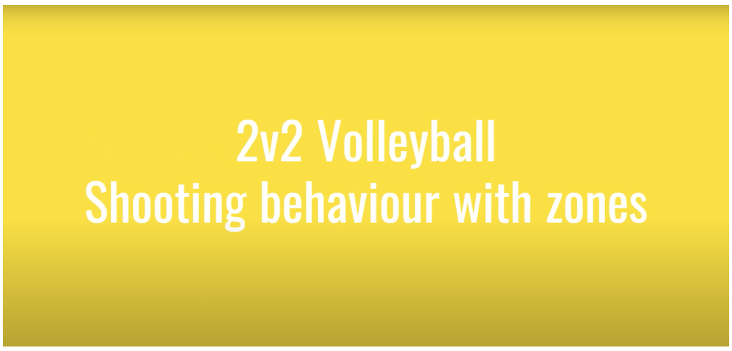


Figure 9.2: Shooting behaviour with zones

Figure 9.3: Passing behaviour with zones

Figure 9.4: Passing and shooting separate. Final Behaviour

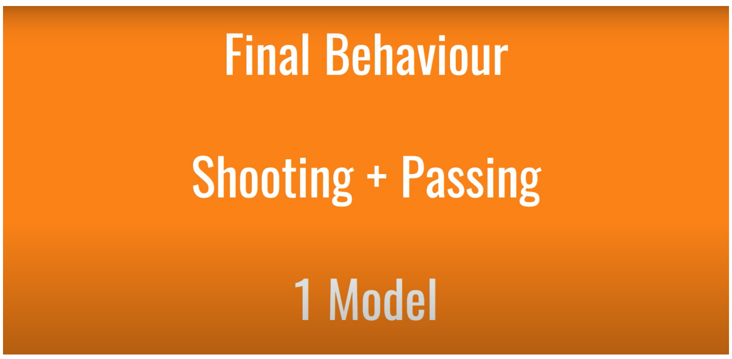


Figure 9.5: Final Behaviour, Passing + Shooting shared policy

Figure 9.6: Final Behaviour, Passing + Shooting separate policies

Figure 9.7: Final Behaviour without jump penalizing, Passing + Shooting separate policies

9.2 Code

The developed code is available in github at: *https://github.com/filipbyberg/2v2_Volleyball_Byberg*

The project package dependencies is found at:

- *https://github.com/Unity-Technologies/ml-agents*
- *https://github.com/CoderOneHQ/ultimate-volleyball*

9.2.1 Hyper parameter

```yaml

behaviors:
  Volleyball:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0005
      beta: 0.003
      epsilon: 0.15
      lambd: 0.93
      num_epoch: 4
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.96
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 120000000
    time_horizon: 1000
    summary_freq: 20000
---
```

Acknowledgements

I want to thank my tutor Cecilio Angulo for his support and guidance through this master thesis. While also allowing me to work with something that is of great interest to me.

I also want to thank Donalys and her family that has welcomed me into their family. Their support has been my main inspiration throughout this master program, and I deeply value all of their support, despite all of us being located in different parts of the world.

Lastly, I will thank my parents Finn and Karen for the support and love I have received.

References

[1] Joy Zhang. An introduction to reinforcement learning. https://coderone.notion.site/An-introduction-to-reinforcement-learning-24ba5acb04a34d22bd3dbe2ef221212a, July 2021. (Accessed on 10/05/2023).

[2] Nicola Dalla Pozza, Lorenzo Buffoni, Stefano Martina, and Filippo Caruso. Quantum reinforcement learning: the maze problem. *Quantum Machine Intelligence*, 4, 06 2022. doi: 10.1007/s42484-022-00068-y.

[3] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020. URL https://arxiv.org/pdf/1809.02627.pdf.

[4] Joy Zhang. Ultimate volleyball: A multi-agent reinforcement learning environment built using unity ml-agents. https://towardsdatascience.com/ultimate-volleyball-a-3d-volleyball-environment-built-using-unity-ml-agents-c9d3213f3064, 2021. [Online; accessed 11-August-2021].

[5] Inc Unity. Unity enterprise, 2023. https://unity.com/products/unity-enterprise?utm_source=google&utm_medium=cpc&utm_campaign=cc_abm_ent_emea_emea-t1_en_aw_sem-gg_acq_br-pr_2023-05_cc-abm-emea-t1-br_cc3022_ev-br_id:7170000109980887&utm_content=cc_abm_ent_emea_aw_sem_gg_ev-br_ct_x_ggl_cpc_kw_sd_all_x_x_3d_id:58700008378954569&utm_term=unity%20game%20engine%203d&&&gclid=Cj0KCQjwmvSoBhDOARIsAK6aV7gKXV02Gd6GcriGXvTsOZCgwEyrfCRIhfKabegWmstIpbSLDvgrLbc wcB&gclsrc=aw.ds [Accessed: 2023].

[6] SA McLeod. Bf skinner: Operant conditioning. *Retrieved September*, 9(2009):115–144, 2007.

[7] Dr.David Abel. Abstract of “a theory of abstraction in reinforcement learning. *Department of Computer Science Brown University*, 1:5, 2020.

[8] Joy Zhang. An introduction to machine learning with unity ml-agents | coder one. https://www.gocoder.one/blog/introduction-to-unity-ml-agents/, August 2021. (Accessed on 10/05/2023).

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801*, page 01290., 2018.

[10] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

[11] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, page 4565–4573. 2016.

[12] OpenAI. Proximal policy optimization. https://openai.com/research/openai-baselines-ppo, July 2017. (Accessed on 10/05/2023).

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707*, page 06347, 2017.

[14] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_1. URL https://doi.org/10.1007/978-3-642-27645-3_1.

[15] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. Revisiting design choices in proximal policy optimization. *CoRR*, abs/2009.10897, 2020. URL https://arxiv.org/abs/2009.10897.

[16] Towards Datascience. Curriculum learning with unity ml-agents, 2020. https://towardsdatascience.com/curriculum-learning-with-unity-ml-agents-c8e7a1aa5415 [Accessed: 2023].

[17] Medium. Transfer learning in reinforcement learning, 2020. https://medium.com/@mauriciofadelargerich/transfer-learning-in-reinforcement-learning-45160ceb31c4 [Accessed: 2023].

[18] T. Bansal, J. Pachocki, Sutskever Sidor, I., and I. Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710*, page 03748., 2017.

[19] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping, 2020.

[20] Dale Hartley. The cobra effect: No loophole goes unexploited | psychology today. `https://www.psychologytoday.com/intl/blog/machiavellians-gulling-the-rubes/202010/the-cobra-effect-no-loophole-goes-unexploited`, October 2020. (Accessed on 10/06/2023).

[21] Andrew Cohen, Ervin Teng, Vincent-Pierre Berges, Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, and Sujoy Ganguly. On the use and misuse of absorbing states in multi-agent reinforcement learning. *RL in Games Workshop AAAI 2022*, 2022. URL `http://aaai-rlg.mlanctot.info/papers/AAAI22-RLG_paper_32.pdf`.

[22] Allan Yashinski. Absorbing markov chains. `https://www.math.umd.edu/~immortal/MATH401/book/ch_absorbing_markov_chains.pdf`, July 2021. (Accessed on 10/05/2023).

[23] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and YI WU. The surprising effectiveness of ppo in cooperative multi-agent games. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24611–24624. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/9c1535a02f0ce079433344e14d910597-Paper-Datasets_and_Benchmarks.pdf`.

[24] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35 - 36th Conference on Neural Information Processing Systems, NeurIPS 2022*, Advances in Neural Information Processing Systems. Neural information processing systems foundation, 2022. Funding Information: This research is supported by NSFC (U20A20334, U19B2019 and M-0248), Tsinghua-Meituan Joint Institute for Digital Life, Tsinghua EE Independent Research Project, Beijing National Research Center for Information Science and

Technology (BNRist), Beijing Innovation Center for Future Chips and 2030 Innovation Megaprojects of China (Programme on New Generation Artificial Intelligence) Grant No. 2021AAA0150000. Publisher Copyright: © 2022 Neural information processing systems foundation. All rights reserved.; 36th Conference on Neural Information Processing Systems, NeurIPS 2022 ; Conference date: 28-11-2022 Through 09-12-2022.

[25] Unity. Gym api documentation - unity ml-agents toolkit. `https://unity-technologies.github.io/ml-agents/Python-Gym-API-Documentation/`, 2023. (Accessed on 10/24/2023).

[26] Unity. Unity - scripting api: Gameobject. `https://docs.unity3d.com/ScriptReference/GameObject.html`, July 2023. (Accessed on 10/15/2023).

[27] Unity. Class academy | ml agents | 1.0.8. `https://docs.unity3d.com/Packages/com.unity.ml-agents%401.0/api/Unity.MLAgents.Academy.html`, July 2023. (Accessed on 10/24/2023).

[28] Unity. Class vectorsensor | ml agents | 1.8.1-preview. `https://docs.unity3d.com/Packages/com.unity.ml-agents@1.8/api/Unity.MLAgents.Sensors.VectorSensor.html`, October 2023. (Accessed on 10/15/2023).

[29] Unity. Class rayperceptionsensor | ml agents | 1.0.8. `https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.Sensors.RayPerceptionSensor.html`, October 2023. (Accessed on 10/15/2023).

[30] Unity. Class camerasensor | ml agents | 1.0.8. `https://docs.unity3d.com/Packages/com.unity.ml-agents%401.0/api/Unity.MLAgents.Sensors.CameraSensor.html`, 2023. (Accessed on 10/24/2023).

[31] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. `https://proceedings.neurips.cc/paper/2020/file/b710915795b9e9c02cf10d6d2bdb688c-Paper.pdf`, 11 2020.

[32] Joy Zhang. Ultimate volleyball machine learning micro-course. `https://joooyz.gumroad.com/l/ultimate-volleyball`, March 2022. (Accessed on 10/26/2023).

[33] Unity. Unity - manual: Prefabs. `https://docs.unity3d.com/Manual/Prefabs.html`, October 2023. (Accessed on 10/26/2023).

[34] Sajjad Saleem. Unity's ml-agents and effective ways of training (2/3) | by we.r.play | medium. `https://werplay.medium.com/unitys-ml-agents-and-effective-ways-of-training-2-3-6eea5c99d1d`, June 2021. (Accessed on 10/26/2023).

[35] Unity technologies. ml-agents/docs/training-configuration-file.md at release_18_docs · unity-technologies/ml-agents · github. `https://github.com/Unity-Technologies/ml-agents/blob/release_18_docs/docs/Training-Configuration-File.md`, 2021. (Accessed on 10/26/2023).

[36] deepanshut041. Soccer twos | reinforcement-learning. `https://deepanshut041.github.io/Reinforcement-Learning/mlagents/05_soccer_twos/`, March 2020. (Accessed on 10/30/2023).

[37] Edouard Geze Yann Bouteiller. Github - trackmania-rl/tmrl: Reinforcement learning for real-time applications - host of the trackmania roborace league. `https://github.com/trackmania-rl/tmrl`, October 2022. (Accessed on 11/09/2023).

[38] Antonio Coronato, Muddasar Naeem, Giuseppe De Pietro, and Giovanni Paragliola. Reinforcement learning for intelligent healthcare applications: A survey. *Artificial Intelligence in Medicine*, 109:101964, 2020. ISSN 0933-3657. doi: https://doi.org/10.1016/j.artmed.2020.101964. URL `https://www.sciencedirect.com/science/article/pii/S093336572031229X`.

[39] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL `http://arxiv.org/abs/1912.06680`.

[40] INTERNATIONAL ENERGY AGENCY. Iea_methodology_emission_factors.pdf. `https://iea.blob.core.windows.net/assets/26122570-1a31-4c25-b8a9-51be776ee6c6/IEA_Methodology_Emission_Factors.pdf`, January 2023. (Accessed on 11/21/2023).