MULTI-AGENT REINFORCEMENT LEARNING: COLLABORATIVE AGENTS IN A MUSEUM ROBBERY

Eleni Evripidou

University of Cyprus
Department of Computer Science

MSc in Computer Science

University of Cyprus

January, 2023

Eleni Evripidou

MULTI-AGENT REINFORCEMENT LEARNING: COLLABORATIVE AGENTS IN A MUSEUM ROBBERY

Eleni Evripidou

Eleni Evripidou

A Thesis
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science
at the
University of Cyprus

Recommended for Acceptance
by the Department of Computer Science
January, 2023

ABSTRACT

Non-Playable Characters (NPCs) in video games play an important role in the development of an immersive video game. Traditionally NPC behaviours are hard-coded, causing human players to easily identify their weaknesses. Generating NPC behaviours with Reinforcement Learning can introduce real time reactions against human players. In this project, we introduce the early results of a multi-agent team's strategy using Reinforcement Learning techniques in a Non-Zero Sum adversarial asymmetric game. We constructed a complex environment that simulates a museum robbery. The successfully trained team is that of robbers, whose goal is to steal valuables from the museum and leave before being noticed by moving security guards and cameras. The robber team consists of two NPCs with different skills. One is called the Locksmith and is tasked with opening doors and the other the Technician tasked with disabling security cameras. We trained each agent with a different policy while providing them with both individual and group reward signals. The agents learn to cooperate while using their skills for both their own and their team's benefit.

Eleni Evripidou – University of Cyprus, 2023

Eleni Evripidou

APPROVAL PAGE

Master of Science Thesis

MULTI-AGENT REINFORCEMENT LEARNING: COLLABORATIVE AGENTS IN A MUSEUM ROBBERY

Presented by

Eleni Evripidou

Eleni Evripidou

Research Supervisor ______________________________

Andreas Aristidou

Committee Member ______________________________

Chris Christodoulou

Committee Member ______________________________

Panayiotis Charalambous

University of Cyprus

January, 2023

DECLARATION

I hereby declare that this dissertation has been composed solely by myself. The work presented was carried out by myself unless stated otherwise by a reference or acknowledgement.

Eleni Evripidou – University of Cyprus, 2023

Eleni Evripidou

ACKNOWLEDGEMENT

This project was submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science at the University of Cyprus and was part of my internship at CYENS under the guidance of Dr Panayiotis Charalambous and the V-Eupnea Group.

I would like to express appreciation towards my supervisors Dr Panayiotis Charalambous and Dr Andreas Aristidou for their guidance, advice, time and resources they provided me with throughout the dissertation project.

I am also grateful for the "Praxandros" Partial Scholarship offered to me by the University of Cyprus. Without it my studies would not have been possible.

Many thanks to my best friend and partner Andreas Michael for his constant encouragement and my cat Nori for cheering me up. Last but not least, I would like to thank my family, for the moral and financial support.

Eleni Evripidou

Contents

1 Introduction **1**

2 Related Work **3**

3 Fundamentals **7**

3.1 Reinforcement Learning 7

Environment and Agent 8

States and Observation 8

Action Space 9

Policy 9

Reward Function and Return 9

Value Function 10

Advantage Function 10

Vanilla Policy Gradient 11

3.2 Proximal Policy Optimization Algorithm 11

3.3 Unity and the ML-Agents Toolkit 13

3.3.1 Unity 13

Collider 13

RayCasting 14

Eleni Evripidou

Rigid Body . . . 14
Navigation System . . . 15
3.3.2 Unity ML-Agents Toolkit . . . 15
Behaviour Parameters . . . 16
Paralellization . . . 16
Curriculum Training . . . 16
4 Implementation **17**
4.1 Scenario . . . 17
4.1.1 Rules . . . 17
4.1.2 Member Abilities . . . 18
Robbers . . . 18
Guards . . . 19
Setbacks . . . 19
4.1.3 Environment Mechanics . . . 20
Map Generation . . . 20
Valuables . . . 21
Security Cameras . . . 22
Guard Behaviour . . . 22
Robber Mechanics . . . 23
Episode Mechanics . . . 23
4.2 Learning Strategy . . . 25
4.2.1 Group Training . . . 25
4.2.2 Actions . . . 26
4.2.3 Observations . . . 27

Eleni Evripidou

4.2.4 Rewards . . . 32
4.2.5 Training process . . . 33
Architecture . . . 33
Vanilla Training Type I . . . 35
Vanilla Training Type II . . . 35
Curriculum Training Type I . . . 35
Curriculum Training Type II . . . 35
4.3 Knowledge Gained . . . 36
4.3.1 Initial Experiments . . . 36
4.3.2 Curriculum Learning . . . 36
4.3.3 Group Training . . . 38
4.3.4 Setbacks . . . 39
5 Results **41**
5.1 Experiment I . . . 41
5.2 Experiment II . . . 43
5.2.1 Environment Changes . . . 46
6 Discussion **47**
6.1 Limitations . . . 47
6.2 Future Work . . . 48
7 Conclusion **50**
Bibliography **52**
Eleni Evripidou

List of Figures

3.1 Interaction Loop between Environment and Agent in RL. 8

3.2 Communication diagram of the ML-Agents Toolkit. 15

4.1 The object representation in the environment. 20

4.2 Layout of the museum map before an episode begins. 21

4.3 Communication diagram between the Group Controller, the agents and the environment. 25

4.4 The different Ray Sensors. Starting from above left we have the Back Rays(pink), Front Rays (red), Valuable Rays (yellow), Security Camera Rays (blue), Robbers & Guards Rays (green). 28

4.5 Layout of environment in the initial experiments. 37

5.1 Cumulative reward received by Locksmith and Technician agents during Vanilla Type I and Curriculum Type I trainings with respect to episodes. We indicate in circles the start of new phases during the curriculum training. 42

5.2 Cumulative reward received by Locksmith and Technician agents during Vanilla Type II and Curriculum Type II trainings. We indicate in circles the start of new phases during the curriculum training. 44

5.3 Cumulative reward received by Locksmith and Technician agents during Curriculum Type II training. 45

Eleni Evripidou

List of Tables

4.1 The rules of the winning/losing scenario depending on the team. 18

4.2 The action space for agents in the team of Robbers. 26

4.3 The categorisation layers for each object 27

4.4 The Ray Sensors and their features used during training. Each Ray Sensor can detect specific objects. A Ray Sensor has a number of rays and a total angle measured between its first and last rays. 28

4.5 The common observations of robber agents are shown in the first three tables. However, each skilled robber observes some extra information about the environment described in the last two tables. The total number of data that the Locksmith agent observes is 64 and that of the Technician agent is 63. 29

4.6 The reward function $R_{1,k}$ for episode k, where t is the current timestep, T is the maximum number of timesteps, T_G is the maximum number of timesteps allowed in the guards' FOV, T_{SC} is the maximum number of timesteps allowed in the security cameras' FOV, T_A is the maximum number of timesteps allowed after the alarm is triggered, T_E indicates that the episode ended at time $t = T_E$ and $d_{GU} = T_E - t_j$. 33

4.7 The reward function $R_{2,k}$ for episode k, where t is the current timestep, T is the maximum number of timesteps, T_G is the maximum number of timesteps allowed in the guards' FOV, T_{SC} is the maximum number of timesteps allowed in the security cameras' FOV, T_A is the maximum number of timesteps allowed after the alarm is triggered, T_E indicates that the episode ended at time $t = T_E$ and $d_{GU} = T_E - t_j$. 34

Eleni Evripidou

Chapter 1

Introduction

Eleni Evripidou

The involvement of Machine Learning (ML) in computer games has led to many interesting Non-Playable Character (NPC) behaviours. Most often NPCs are not particular intelligent and can easily be defeated by human players in combat games. Adding interesting traits to the NPCs can lead to more intuitive and unexpected behaviours that can positively impact the experience of a computer game player.

Traditionally NPCs in games are hard-coded using several methods of directed graphs like Finite State Machines, Decision Trees and Behaviour Trees. These methods can be very limiting to the agents' behaviours, especially when NPCs are the opponents or bosses that human players must defeat. Most often human players learn how to defeat NPCs by identifying traits, blind spots and weaknesses that are caused by the limitations of their hard-coded behaviour.

Games have always played a significant role in advancing Artificial Intelligence (AI) technology. In 1992 TD-Gammon was trained using self-play to play Backgammon [1] at a level slightly lower compared to that of the top human players. In 1997 the Deep Blue IBM computer beat the world chess champion [2] and in 2013 Deep Mind developed an algorithm that plays Atari that surpassed a human expert player in some

levels of the game [3]. Deep Mind has followed this success with the development of AphaGo, the first computer program to defeat the world Go champion in 2016 using tree search [4]. The algorithm continued to expand with the development of AphaGo Zero, a program trained without human knowledge and which exceeded the level of AphaGo [5]. Lastly, they generalised their algorithm developing AlphaZero to play Chess, Shogi and Go with superhuman performance [6].

These breakthroughs focused on games like Chess, Go and Backgammon, which are 1v1 games with discrete, deterministic and fully observable environments. On the other hand, video games have dynamic, continuous and complex environments that can allow multiple players to play simultaneously. In 2019 both Deep Mind and OpenAI concurrently worked on training agents in real time strategy video games. Deep Mind created AlphaStar which was successfully rated at Grandmaster level in StarCraft II [7]. At the same time OpenAI developed a model named OpenAI Five that was the first one to defeat the world champions at an esports game, Dota 2 [8]. In 2020 OpenAI used a multiagent environment in Hide and Seek that resulted in some bizarre behaviours, while creating a self-supervised autocurriculum [9].

Eleni Evripidou

In this project we used Reinforcement Learning (RL) methods and specifically the Proximal Policy Optimisation (PPO) algorithm to train a team which is constructed by multiple agents, each with different skills and who must cooperate to achieve their goal. The agents use their skills to help both themselves and their teammates. We show that using different policies for each agent can lead to a cooperation, and generation of a strategy that prioritises the whole team.

In chapter 2 we summarise the related research work that focused on AI in video games. In chapter 3 we establish the fundamentals of Reinforcement Learning, PPO algorithm and how the ML-Agents Unity Tool works. Furthermore, in chapter 4 we focus on the technicalities and implementation of our work. In chapters 5 and 6, we evaluate our results and discuss future work and limitations.

Chapter 2

Related Work

Our goal is to generate NPCs which have more complex behaviours that are not scripted, such that players feel like they are playing against humans. We focus on multiplayer real time video games with complex environments and strategies.

Deep Mind's AlphaStar has achieved complex strategies in StarCraft II [7] reaching grandmaster level. StartCraft II is a 1v1 game, where each team is composed by different units which are controller by the player. Hence all units are controlled by the same brain. Deep Mind used multiple ML techniques. For example, it uses supervised learning techniques to imitate human replays, in order to construct a statistical function that decides the builder order. In addition, it implements RL techniques and specifically advanced actor-critic methods to train the agent to win. Actors send sequences of observations, actions, and rewards over the network to a critic that updates the parameters of the training agents. It used the zero-sum game idea for assigning rewards signals. Specifically, the agent receives +1 fora win, 0 for a draw and -1 for losing without the addition of a discount factor. Exploration issues and the sparse nature of the rewards were a challenge. To overcome these issues, the authors used human data to help with exploration and preserve strategy throughout training. Furthermore, they used fictitious self play, which is when the main agent plays against older versions

Eleni Evripidou

of themselves. In addition, they created two types of agents, the main exploiters and the league exploiters, that play against the current iteration of the main agent. Their job is to identify the main agents' exploits. Hence, by playing against these kind of exploiters, the main agent learns to defend its weak points.

Another stategy based video game is Dota 2, which is a multiplayer game. Each team consists of five individual players that control a character with specific skills. OpenAI Five is the first program to defeat the world champions [8] in 2019. The environment has similar challenges to StartCraft II such as long time horizons, partial observability of the environment, complicated rules, high dimensionality of observation and action spaces and continuous state-action spaces. The team of OpenAI used self play tactics, a distributed training system and tools to resume training after making changes such that training does not restart from scratch. OpenAI Five team is controlled by five separate replicas of the same policy. The policy is trained using Proximal Policy Optimization (PPO). Though each player has different skills such as strength, spells to cast etc, their observation and action space size is the same. In order for the agent to decide which action to take, they observe the type of character they control. Since the game is very complicated, the authors also included some hard-coded features to make decisions. To achieve this extraordinary performance they used a distributed training system and a technique called surgery to allow for continual training that lasted 10 months.

In 2020, OpenAI released the results of a multi-agent competition for the game Hide and Seek using PPO and Generalized Advantage Estimation (GAE) [9]. They found that agents create a selfsupervised autocurriculum, that includes distinct stages of different strategies. Their implementation contained two teams, the hiders and the seekers. The role of the hiders was to avoid the line of sight of the seekers, and the role of the seekers was to keep the hiders in their vision. The agents are trained using self-play regardless of their different roles (hider, seeker). During training, a team created a strategy to always win, pressuring the opposing team to change their strategy. This created distinct training stages that can be thought of as the stages of curriculum training, where the difficulty of the environment is gradually increased such that the agents learn how to

Eleni Evripidou

solve more and more complex tasks.

As we have seen in previous works, the approach is to train one policy using self-play. However, this is dictated by the type of game. All described games are adversarial Zero-Sum games, meaning games in which a player's gain is equivalent to the opponent's loss. In addition, these kinds of games can be categorised as symmetric or asymmetric. Symmetric games are games such as tennis, football, backgammon, GO, chess etc. For example, in football, each team's role is to attack their opponents by scoring a goal and defend their team by preventing opponents to score a goal. In these type of games we can use self-play methods to train the agents to play against their past selves. We can also use this technique for asymmetric games where each team has different roles, like Hide and Seek. While training a team playing against a past self of its opponent team that gradually changes such that it plays against an opponent team with the highest currently available level.

We would like to examine Non-Zero Sum adversarial asymmetric games, like Valorant, where each player is controlled by a different human individual, with different playing styles. We want to understand what strategy a team of agents, with different skills, develops to win against an opponent team with a different role. In our project, we use PPO to train the agents provided by the Unity ML-Agents Toolkit. We use different policies to train each agent with a different skill, since the observation and action space varies in size depending on the agent. Our work is a toy example that includes a complex environment and encourages agents to generate a strategy. We produce a promising result which shows that a team of agents with different skills and trained using different policies, develop a cooperative strategy to win while prioritising their team their team. We decided to examine this by introducing the scenario of a museum robbery. We have developed a new environment in Unity from scratch that simulates a museum and is equipped with security cameras, furniture, gates, and valuables. The two teams competing against each other are the team of robbers and the team of guards. However, at this moment, our results only refer to the team of robbers, due to unexpected challenges. The team consists of two individual agents, the Locksmith and

Eleni Evripidou

the Technician. The Locksmith opens doors and the Technician can disable security cameras. We show that the two members discover a way to use their skills to help themselves and their teammates, while avoiding enemies to complete their goal.

Eleni Evripidou

Chapter 3

Fundamentals

3.1 Reinforcement Learning

Machine Learning (ML) consists of different types of learning. Reinforcement Learning (RL) is a type of learning where the agents receive a reward or a penalty signal depending on their actions and the state of the environment. Hence, the agent is learning with a critic and not with a teacher like in Supervised Learning. In this type of learning the agent is not being told which actions to execute directly but it discovers which actions maximise its reward through trial and error. RL has its roots in the natural way of animal learning psychology in the real world. Biological brains are hardwired to avoid actions that cause pain and seek actions that cause pleasure. For example, dogs can be trained to do tricks over time but receiving treats which act as a reward signal [10]. RL can be used in dynamic environments and problems that are similar to real-world scenarios.

Eleni Evripidou

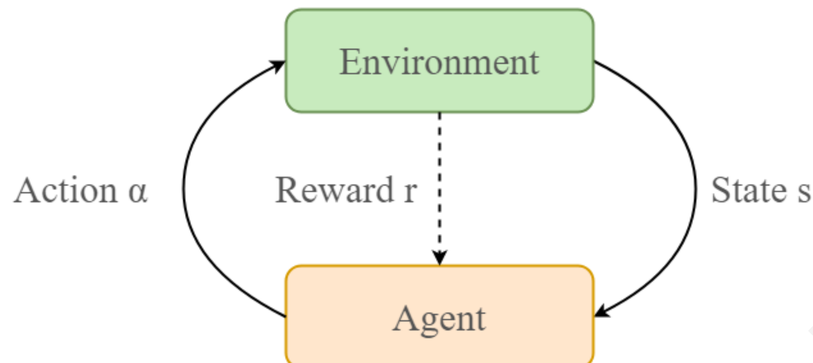

Figure 3.1: Interaction Loop between Environment and Agent in RL.

Environment and Agent

The key components in RL are the environment and the agent. Their interaction loop can be seen in the graph 3.1. The environment represents the world in which the agent lives and can interact with. It acts as a critic to the actions of the agent depending on its current state. Specifically, the agent observes the state s of the environment fully or partially. It is given as a sensory signal and the agent acts upon it. The actions a of the agent change the state of the environment. The environment then criticises these actions and gives a reward or a penalty signal r to the agent. The goal of the agent is to maximise its cumulative reward. It must be noted that the environment does not change only from the the agents' actions but can also change independently due to other parameters.

Eleni Evripidou

States and Observation

The state s contains all the information about the state of the environment. However, the agent can observe this state fully or partially. If the agent can observe the complete state s of the environment then the environment is fully observed. Otherwise, if there is information hidden from the agent, then the environment is partially observed. The observations can be represented as a vector with real numbers, a matrix or even high-

order tensors depending on the problem.

Action Space

The action space is a set of valid actions that the agent can take in a given environment. When the agent can choose between a finite number of available actions, the action space is discrete. On the other hand, an action space can be continuous when the actions are real-valued vectors. For example, a discrete action space is Turn Right, Turn Left and a continuous action space can be a real number that ranges between [-1,1].

Policy

The policy π is the Neural Network (NN) that accepts observations as input and provides actions as an output. It is a decision making function $\pi = \pi(a_t|s_t)$ that maps the state of the environment s_t to the action a_t. The policy can be considered as the brain of the agent.

Reward Function and Return

The reward r_t that the agent receives from the critic is the result of the reward function R. This function depends on the current state s_t, the next state s_{t+1} and the current action taken a_t,

$$r_t = R(s_t, a_t, s_{t+1}) \tag{3.1}$$

The agent's goal is to maximise the cumulative reward over time. The return G_t is called infinite-horizon discounted return and is the weighted sum of all received rewards after the time step t,

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{3.2}$$

Eleni Evripidou

where $\gamma \in (0, 1)$ is a discount factor that represents how valuable the same reward is in the future. For example, getting \$100 the next day is more valuable than getting the same amount 10 years from now. If the discount factor $\gamma \to 0$ then the agent chooses an action that maximises the immediate reward but this can decrease the cumulative reward. On the other hand, when $\gamma \to 1$ it gives the agent foresight, such that they will choose actions that can affect their future.

Value Function

The value function $V^{\pi}(s)$ computes the expected value of the discount return when using the current policy π from this point onward, starting from state s. Hence, it is an estimation of the cumulative reward, assuming that the agent always uses the current policy to decide what action to take. Note here that this definition is for on-policy algorithms, so algorithms where actions are always selected according to the current policy.

Advantage Function

The advantage function $A^{\pi}(s, a) = \hat{A}_t$ can be computed via

$$A^{\pi}(s, a) = G_t - V^{\pi}(s_t) \tag{3.3}$$

and describes how much better the selected action α when in state s is compared to the baseline, when using policy π from time instance t onward. An action a is considered better if it leads to a higher return compared to the expected value $G_t > V^{\pi}(s_t)$.

Eleni Evripidou

Vanilla Policy Gradient

The Vanilla Policy Gradient method can be used to train a policy network π. The goal of the policy gradients is to increase the probability of action sequences that lead to higher returns happening again in the future and to decrease the probability of action sequences that lead to lower returns. The available actions have a probability with which they are selected, hence during training these probabilities are adjusted using a gradient based method such that to increase the probabilities of actions that lead to positive rewards. This is achieved by differentiating the optimization objective

$$\hat{\mathbb{E}}_t \left[\log \pi(a_t|s_t) \hat{A}_t \right] \tag{3.4}$$

If $\hat{A}_t > 0$ then the gradient is positive and these actions' probability is increased. If $\hat{A}_t < 0$ the gradient is negative hence the likelihood of these actions happening in the future is decreased.

3.2 Proximal Policy Optimization Algorithm

The Proximal Policy Optimization or PPO algorithm is an on-policy algorithm that uses stochastic gradient methods delevoped by OpenAI [11]. PPO is based on the TRPO algorithm [12]. Both try to update the policy in the best possible way without significantly deviating from from the current policy and destroying it. PPO outperforms TRPO, is simpler to implement, is data efficient and has equal stability and reliability to TRPO. PPO uses an objective function to achieve the regularisation of the policy update while reusing data.

The standard PPO has a clipped objective function L^{CLIP} given by

$$L^{CLIP} = \hat{\mathbb{E}}_t \left[\min \left(p_t \hat{A}_t, \text{clip}(p_t, 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \tag{3.5}$$

Eleni Evripidou

where $p_t = \dfrac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}$ and ϵ is a small number that defines how far the new policy can move from the current one. The first term inside the min() function is the objective function of TRPO. If $p_t > 1$ then the action is more likely to happen in the new policy rather in the old policy. Similarly, if $0 < p_t < 1$ the action is less likely to happen in the new version as opposed to the older version. This, combined with the advantage function, does the same thing as the Vanilla policy gradient, hence choosing actions that yields to higher positive return. The second term in the min() function is a clipping operation ensuring that the new policy will not drastically change compared to the older policy. In the case of $A > 0$ and $p_t > 1 + \epsilon$, the objective function L flattens to limit the effect of the gradient update. Similarly, in the case of $A < 0$ and $p_t < 1 - \epsilon$, the objective function L flattens to stop overdoing the update. Basically, this limits the action probabilities such that they do not change drastically from only one update. This can easily happen because the advantage function is very noisy, and we want to avoid destroying our policy from a single estimation. The existence of the objective function allows the policy to update for multiple epochs in minibatches of the sampled data rather than update per one sampled data.

Algorithm 1 PPO

```
for iteration = 1, 2, ... do
  for actor = 1, 2, ...N do
    Run policy π_old in environment for T timesteps
    Compute advantage estimates Â_1, .., Â_T
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size M ≤ NT
  θ_old ← θ
end for
```

The algorithm's pseudocode can be seen in Algorithm 1 as described by the authors in [11]. Each iteration has N actors which collect data from T timesteps by running the current policy in the environment. The policy is then optimised using the objective function L and applying gradient descent on a minibatch of sampled data for K epochs.

This algorithm was chosen to train the agents in our scenario since it is an on-policy algorithm, thus it does not suffer from convergence issues and can support paralleli-

Eleni Evripidou

sation as opposed to off-policy algorithms. Additionally, PPO is simple to implement and has shown in literature that it is robust and data efficient in video games like Atari [3], Dota 2 [8] and Hide-Seek [9] compared to other on-policy algorithms. In addition, PPO can be used in environments that have both discrete and continuous spaces and our scenario contains both, as we will describe in Section 4.1. Furthermore, it can be easily adjusted to our need by tweaking some hyperparameters.

3.3 Unity and the ML-Agents Toolkit

Our project includes an environment with moving agents, moving obstacles and complex interactions between them. Unity was used in this project to create the 3D environment since it has embedded tools that are used to develop the mechanics of objects.

3.3.1 Unity

Unity is a game engine with embedded useful tools that allow users to create a 2D or 3D environments for video games, films or even industrial usage. It is designed for simulating worlds with sophisticated physics and it allows to create complicated interactions between objects. In the following section we introduce some of the tools used in this project for the development of the environment [13].

Collider

Colliders can be attached as components to objects. Unity includes different shape colliders such as sphere, box and capsule. Additionally, it allows users to create their own colliders by attaching the Mesh Collider component to an object. This builds a collider in the shape of a given mesh. This enables a more accurate collision detection between objects with complicate meshes. Depending on the settings, colliders can

Eleni Evripidou

collide simulating physical phenomena such as a ball bouncing of the wall. The other option is for the physics engine to ignore collisions so colliders can intersect. This can be used to trigger an event when an intersection takes place. Furthermore, a name can be assign to an object with a collider for identification reasons and objects can also be categorised into different layers.

RayCasting

Unity allows to shoot rays from a point in space towards a direction or an end point. A ray can intersect with objects that have a collider. There is the option to choose specific layers of objects that a ray can intersect with. In addition, when a ray intersects with a collider, it can detect a specific object using its identification name. A common usage of raycasting is to simulate the field-of-view (FOV) of an agent. There are two ways to simulate the FOV. One way is to shoot a number of rays starting from the centre of an agent to different directions in a 2D plane. Then it can be detected if a ray intersects with specific colliders in the world. Another way to do so, is to shoot only one ray from the center of the agent to the desired collider's position and check if an intersection took place only when the ray is inside the agent's FOV.

Rigid Body

For an object in unity to interact with other objects with a physically realistic way, a Rigidbody component must be attached to it. Combined with a collider it enables collision to take place. If no Rigidbody exists in a collider then collisions and intersections as described above cannot be triggered. Additionally, this component allows to apply forces to objects like gravity forces, impact forces or just movement forces. There are different methods to apply a force to an object depending on the chosen settings. For example, a force can be applied to an object so that its velocity is changed directly to the desired one rather than adding an impulse force $mass * acceleration$.

Eleni Evripidou

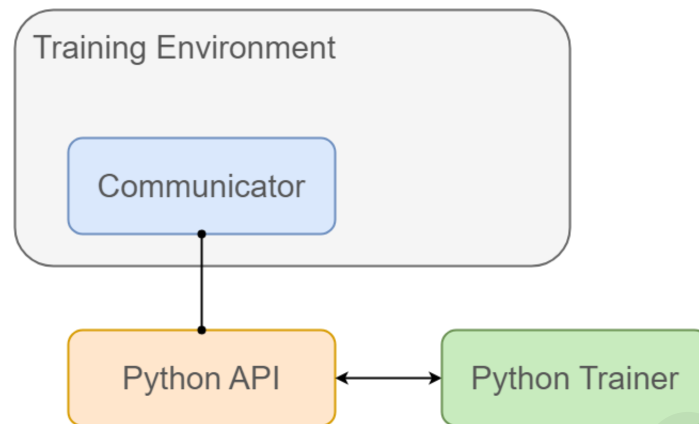

Figure 3.2: Communication diagram of the ML-Agents Toolkit.

Navigation System

Unity offers a tool to create navigation areas that characters can walk in. To navigate characters to target points this tool uses path-finding techniques to choose the path with the lowest cost.

3.3.2 Unity ML-Agents Toolkit

Unity ML-Agents Toolkit was developed in 2020 as a general platform allowing users to use popular reinforcement learning algorithms to train intelligent agents in complex environments [14]. The ML-Agents Toolkit is open source and available in GitHub [15]. This toolkit provides the user with build-in classes and methods to create a training environment in Unity. The training environment includes all the agents to train and objects to interact with. Python Trainer is responsible to train the agents. It exchanges information with the Python API which can communicate with the training environment through the External Communicator (See figure 3.2). The Training Environment sends the observations of the agent to the Python Trainer, and the Python Trainer sends back the chosen actions. Both Python API and Python Trainer are not part of Unity.

Eleni Evripidou

Behaviour Parameters

Behaviour parameters are the parameters of the policy. There are different types of sensors to add to the observation space such as Vector Sensors, Ray Sensors, Grid Sensors or Camera Sensors. The action space can be chosen to be either continuous, discrete or both. Discrete actions are called branches and each branch can have a different size.

Paralellization

To speed up training, a training environment can be duplicated many times in a Scene. The design of ML-Agents allows for each training environment to sent data to a common training buffer which updates the agent's policy. Hence, agents who share the same policy and exist in different training environments would share their experiences during training.

Curriculum Training

Curriculum Learning is a way to train an agent to complete a complex task by introducing gradually parameters that make the environment more difficult. For example, for an agent to learn to jump a high wall, we can introduce at first an environment with no wall, and gradually increasing the height of the wall [16].

Eleni Evripidou

Chapter 4

Implementation

4.1 Scenario

The scenario we are testing in this project is a museum heist. There are two teams: the team of robbers and the team of guards. The two teams have different goals, guards must defend the museum and its collectable objects and robbers must steal the collectables without being noticed by the guards. In addition, individual agents have different skills. We constructed the team of robbers to consist of two types of robbers: the locksmith whose skill is opening doors and the technician whose skill is disabling security cameras. Likewise, the team of guards consists of a patrolling guard and a guard responsible to look the security camera feed.

4.1.1 Rules

In this scenario a win for a certain team is not equivalent to a loss of the other. The team of robbers win only if they successfully steal and get the valuables to the goal area without setting the alarm off. If the robbers hesitate to steal and do not enter the museum, they lose. The team of guards win only if they set the alarm off when

Eleni Evripidou

	Win	Lose
Robbers	Steal valuables and arrive to the goal area without alarming the guards.	Do not make an effort to steal. Make an effort to steal but alarm the guards.
Guards	The night ends and the valuables are still in place. A heist has taken place and the alarm is triggered.	A Heist has not taken place but the alarm is triggered.

Table 4.1: The rules of the winning/losing scenario depending on the team.

the team of robbers have made an effort to steal, otherwise if a robbery has not taken place and the guards trigger the alarm, they lose.

4.1.2 Member Abilities

As discussed above, a member of a team has a special ability that is useful to the team. However, despite the variability in skills of individuals, some actions can be executed by every type of member in a specific team.

Robbers

The team of robbers is composed by two types of robbers, the Locksmith and the Technician. Both types of members can collect a valuable despite their special skills. This is coded in the mechanics of the game. On the other hand, only a Locksmith can unlock a door and only a Technician can disable a security camera.

Eleni Evripidou

Guards

The team of guards consists of two types of guards, the Patrolling Guard and the Security Guard. Both types of members can trigger the alarm. On the other hand, only a Security Guard can receive feedback from the Security Cameras and only a Patrolling Guard can patrol the area.

Setbacks

We initially aimed to train both teams using self-play. In our initial experiments we wanted to make sure that the observations and rewards given during training would help the agent use their skills and achieve their goal, since the environment and mechanics were complex. Hence, we trained separately the Locksmith and the Technician agent. This set us back due to issues with the tuning of observations and rewards as well as with the mechanics of the environment. For further details we suggest looking at section 4.3. Therefore, we concentrated on one team, that of robbers. Our goal is to train the team of robbers to try and steal as many valuables as they can from the museum without getting noticed by the guards and security cameras. Therefore, the team of guards is not trained in this implementation. The guards' behaviour is hard-coded and explained in the following section. In this implementation, we would like to comprehend how the individual team members can collaborate to achieve their goal and learn to use their skills to help their-selves and their teammates, without receiving direct rewards.

Eleni Evripidou

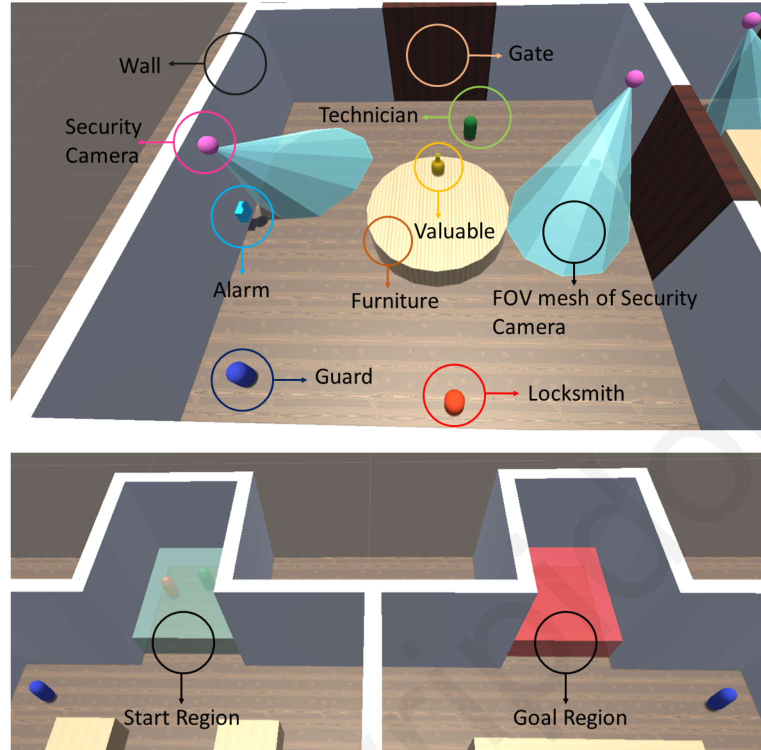

Figure 4.1: The object representation in the environment.

4.1.3 Environment Mechanics

Map Generation

The museum that has been constructed is composed of four rooms each labelled as room i, where $i = [1, 4]$. Each room is connected to the next room by a locked gate. In the beginning of an episode a start region and a goal region spawn connected to the museum. To force robbers to pass through all four rooms before reaching the goal region we have created an algorithm for the creation of a map. An easy implementation is to connect the rooms with gates, as seen in the figure 4.2. Each gate g connects the rooms $j = 1 + (g \mod 4)$ and $k = 1 + ((g + 1) \mod 4)$ where $g = [0, 3]$. At the start of an episode, a gate g is disabled, hence rooms j and k will be connected to either the start region or the goal region. The choice of whether the start region will be connected to room j and the goal region to room k is decided using a parameter which indicates a clockwise or anti-clockwise path. The algorithm was implemented such that the user

Eleni Evripidou

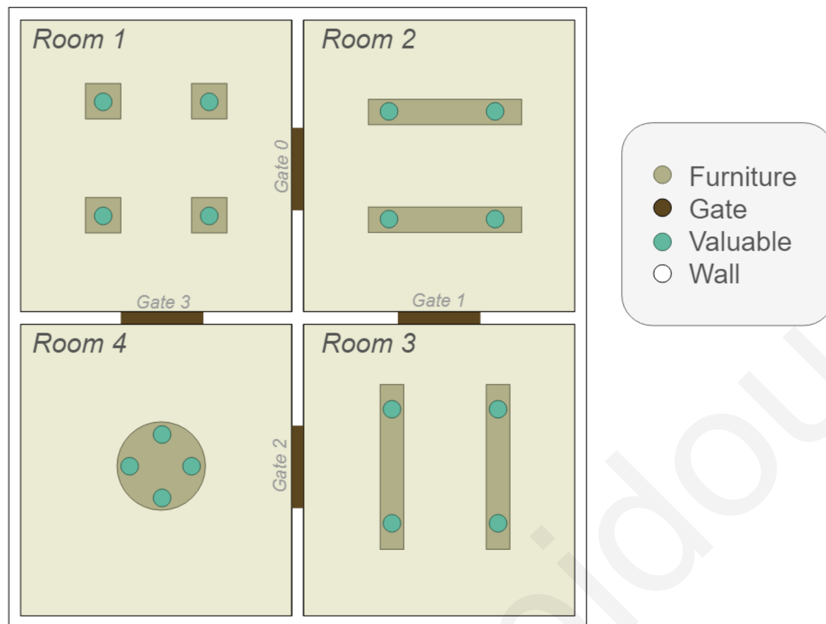

Figure 4.2: Layout of the museum map before an episode begins.

can choose to generate a map randomly, by generating a random gate to disable and a random directional path to follow or by a specific input of the user. For example, if user would like to disable gate 2 that connects the rooms (3,4) and would like a clockwise path. This leads to room 4 to be connected to the starting region and room 3 to the goal region.

Valuables

A valuable is placed in each room. Before the start of an episode, there are four valuables in each room as can be seen in figure 4.2. In the beginning of an episode, one out of the four available valuables in each room is randomly chosen and the rest are removed from the environment. All valuables have a box collider attached to them that can intersect with other colliders.

Eleni Evripidou

Security Cameras

The security cameras can detect robbers. To enable detection we created a cone like shaped mesh collider. We reference this collider as the FOV mesh of a Security Camera, because it represents the field of view of the camera. These colliders can intersect with other colliders, but we are interested in their intersection with the colliders of the robbers. Whenever they intersect with a robber, a ray is cast, starting from the security camera and ending at the feet of the robber. If this ray does not intersect with any obstacles, like furniture, then the robber is detected by the security camera, otherwise they are not. There are two security cameras in each room, meaning eight in total. At least one security camera is placed above a gate. Each camera rotates with an adjustable speed. A security camera can trigger the alarm when it detects a robber for a specific amount of time T_{SC}. We assumed that the security camera feeds are watched by a ghost guard, that does not exist in the environment. Hence, when this ghost agent looks at the feed, and notices peculiar movements for dt_{s_i} time, detected by security camera s_i they will trigger the alarm when $\sum_{s_i} dt_{s_i} > T_{SC}$.

Guard Behaviour

As explained above, in this implementation we hard coded the mechanics of the guards' behaviour. There is one guard in each room and every guard follows their own path. The path is created by inserting checkpoints in the navigation area. A guard can navigate from a checkpoint to another using the Navigation System provided by Unity. Along their path, guards can detect robbers inside their field of view. We implemented this by creating a FOV with an adjustable radius ρ and full angle ϕ. We cast a ray from the center of guard $\vec{c}_g$ to the center of robber i, $\vec{c}_{r_i}$. If the distance $d = |\vec{c}_g - \vec{c}_{r_i}| < \rho$ and $\arccos(\hat{c}_g \cdot \hat{c}_{r_i}) < \frac{\phi}{2}$ then robber i is detected by the guard. The guard can trigger the alarm when they detect a robber for a specific amount of time T_G. Since the guards can communicate with other guards, if a guard A detects a robber for dt_A time period and guard B for dt_B and $dt_A + dt_B > T_G$, then the final guard who detected a robber,

rushes to the alarm to trigger it.

Robber Mechanics

Robbers can interact with different objects in the scene. All types of robbers can interact with valuables as stated above. We allow robbers to collect a valuable only when their sphere collider intersects with the collider of a valuable. Then is up to the robber to decide whether to pick up the valuable or not. On the other hand, only a Locksmith agent can open a gate. This can happen when a different sphere collider assigned only to the Locksmith agent intersects with a gate collider and the gate is not permanently closed. The gate opens only if the Locksmith decides to open it. Similarly, the Technician agent has its own capsule collider to disable cameras. The mechanics of these are the same, but the Technician can only disable cameras that are placed in the same room as them.

Episode Mechanics

In the beginning of an episode the team of robbers spawns in the start region. Each agent spawns in a random position and with a random orientation. The moment both agents exit the starting region, a wall is spawned in order to block their way back to the start region. In addition, when a Locksmith agent acts to open a gate, the gate opens with a delay of 1 seconds, and remains open for 14 seconds before closing again. The Locksmith can open this gate again only if the Technician has not moved forward with them into the new room. As soon as both agents move to a new room, the gate connecting the prior and current rooms is permanently closed. This means that the Locksmith is not able to open the gate again. We implemented this to help the robbers find their way to the goal, rather than wonder around in previous rooms. Furthermore, as soon as the Technician decides to disable a security camera, the camera is no longer working and it remains as such for 30 seconds before turning back on. Lastly, when

Eleni Evripidou

the alarm is triggered, the episode ends as soon as T_A timesteps have passed. The parameter T_A is a time period from the instance the alarm has been triggered to the end of the episode.

An episode lasts for $T = 10000$ timesteps. This means that if nothing triggers the end of the episode before this time passed, the episode is interrupted at $t = T_E < T$. The episode can end either when all robbers reach the goal region, when the alarm is triggered at t_i and $t = t_i + T_A < T$, or when $t = T$ has been reached. Hence, when a single agent A reaches the goal region at timestep t_A, the episode does not end. They remain "alive", but they are unable to take any actions, while they are still receiving rewards. As soon as their teammate B reaches the goal at timestep t_B, the episode ends. The time difference $dt_{GU} = t_B - t_A$ is the total time period that the agent A has left the agent B alone.

Additionally, we created two different environment types referred as Environment Mechanics Type I and II. We constructed Environment Mechanics Type II to be a more strict against robbers. The difference between the two types is the mechanism that allows guards to detect robbers inside the starting region and the guards' FOV parameter value ϕ.

Specifically, in the Environment Mechanics Type I, guards cannot detect robbers that are inside the starting region. This applies to Environment Mechanics Type II before any robber exits the start region. However, as soon as an agent exits the staring region, we allow guards to detect any agent that is lurking inside the starting region. This mechanism was added to discourage robbers taking shelter in the start region, in case only one robber has exited. For the Environment Mechanics Type I this is not a requirement since the reward function is different, as we will explain in section 4.2.4. Furthermore, the value of the guards' FOV parameter in the Environment Mechanics Type I is set to $\phi = 90°$, and in the Environment Mechanics Type II is set to $\phi =150°$. All other details remained unchanged in both types.

Eleni Evripidou

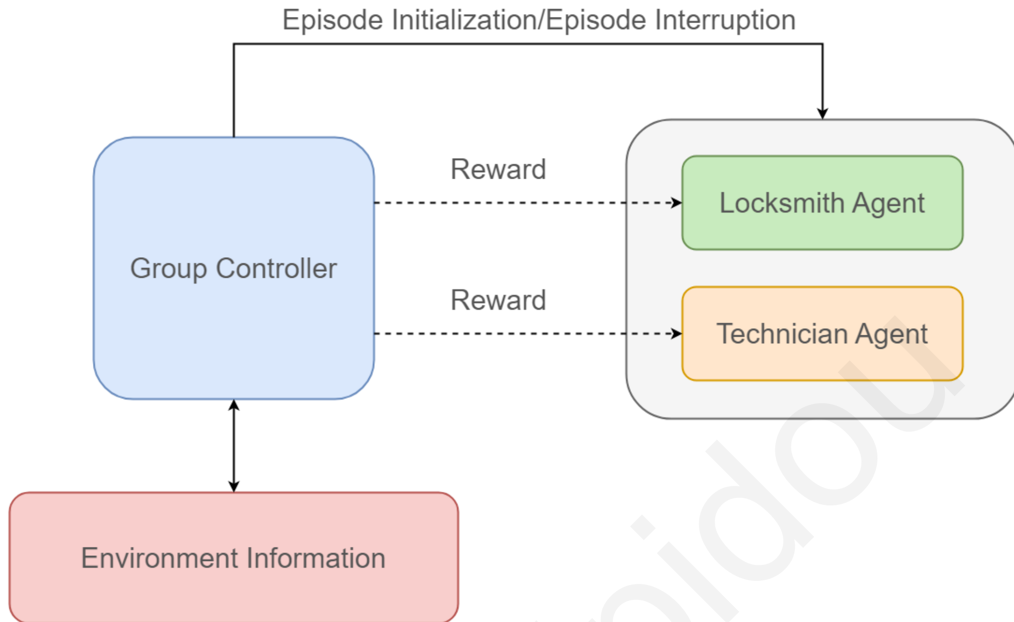

Figure 4.3: Communication diagram between the Group Controller, the agents and the environment.

4.2 Learning Strategy

4.2.1 Group Training

The Unity ML-Agent Toolkit provides the user with the class called Agent that includes methods to start an episode, end an episode, add observations and reward signals. Normally an instance of this class is attached to the agents. Agents with different policies require different instances of the class Agent. Some refer to this as the brain of the agent. Since our scenario requires agents with different skills we created different brains for each one that work independently. Hence, there are instances that the start or the end of an episode do not occur at the same time. Another challenge are group reward signals, which are signals that both agents must receive for a state of the environment. To overcome these challenges, we have created a Group Controller that manages all this information (Figure 4.3). The Group Controller exchanges information with the environment. One of its core functionalities is to initialise the environment. Furthermore, during training, the Group Controller decides whether to initialise or interrupt

Eleni Evripidou

All types of Agents	
Action	Available Values
Vertical Movement	[-1,1]
Horizontal Movement	[-1,1]
Rotation	{Do not rotate, Rotate clockwise, Rotate anti-clockwise}
Pick Up Valuable	{Do not pick up Valuable, Pick up Valuable}

Technician Agent	
Action	Available Values
Disable Security Camera	{Do not disable Security Camera, Disable Security Camera}

Locksmith Agent	
Action	Available Values
Open Gate	{Do not open gate, Open gate}

Table 4.2: The action space for agents in the team of Robbers.

an episode. Hence, at any given moment in time the Group Controller simultaneously signals all different brains that an episode has started or ended. Lastly, the Group Controller is in charge of assigning the reward signals to the agents, either individually or collectively.

4.2.2 Actions

The available actions that the agents can take are both discrete and continuous. For the team of guards, both types of agents have two continuous actions and three discrete action branches. The available actions of the Locksmith agent and Technician agent are described in table 4.2.

The 2D movement of all types of agents includes the same mechanics. We followed the movement blueprint for a First-Player (FP) game type. This blueprint includes horizontal movement using keys A and D and vertical movement using keys W and S. The rotation movement is controlled using a mouse and is independent to the horizontal and vertical movements. We set the horizontal and vertical movements to be float numbers in the range $[-1, 1]$ since this is a common practise followed for the "WASD"

Eleni Evripidou

keyboard controllers. These values are translated to a direction in 2D space. A force in this direction is added to the agent, which changes the value of the agent's velocity directly. Since the forward direction of a player does not coincide with their velocity direction in FP games, the rotation must be controlled independently. We chose the rotation set to include three discrete values. This value rotates the agent clockwise or anticlockwise with a turning speed that it is set by the user. Hence, only the rotation affects the orientation of the agent.

Furthermore, both types of agents have the ability to pick up the collectable items in the museum. The available options are simulated using a boolean value. For an agent to successfully pick a valuable up, their sphere collider must be inside the valuable's box collider and additionally they must choose the action "Pick up Valuable".

Similarly, the specialised skills that each type of agent has is simulated with a boolean value. The Locksmith agent can open a gate only if they choose the action "Open Gate" and their sphere collider intersects with the box collider of the gate. However, the gate must not be open or permanently closed for this to work. Likewise, the Technician agent can disable a security camera only if they choose the action "Disable Camera", their capsule collider intersects with the sphere collider of the security camera and the security camera is on.

4.2.3 Observations

Layer 1	Layer 2	Layer 3	Layer 4
Wall Gate Valuable Furniture Start Region Goal Region	FOV mesh of security cameras	Robber	Guard

Table 4.3: The categorisation layers for each object

Eleni Evripidou

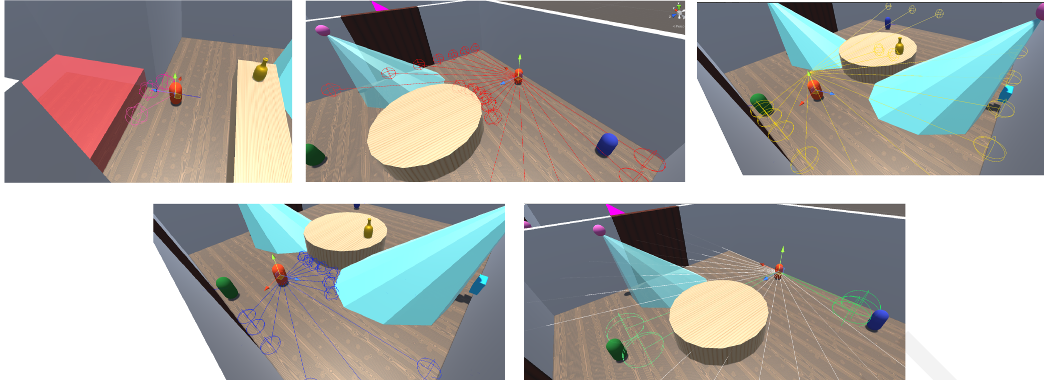

Figure 4.4: The different Ray Sensors. Starting from above left we have the Back Rays(pink), Front Rays (red), Valuable Rays (yellow), Security Camera Rays (blue), Robbers & Guards Rays (green).

We included two types of observation sensors, Ray and Vector Sensors. Ray Sensors are used to detect multiple different types of objects in the environment. Vector Sensors are used to give direct information about specific data. The observation space for each type of robber is slightly different due to their different skills. However, both agents have the same Ray Sensors described in table 4.4.

To explain our approach to create the Ray Sensors, first we define the layers that each object is categorised into, in table 4.3. A ray intersecting with a collider in a specific layer can detect an object only if it can identify its name.

Moreover, it is best to minimise the observation space by using fewer ray sensors. However, it is necessary to include multiple ray sensors to detect all desired objects since each ray intersects only with one object. For example, if object A with height h_A

Ray Sensor Name	Detectable Objects	(# of rays, angle)
Back Rays	Walls, Gates, Furniture, Goal area	(3,90°)
Front Rays	Walls, Gates, Furniture, Goal area	(19,180°)
Valuable Rays	Valuables	(15,180°)
Security Camera Rays	FOV Mesh of Security Cameras	(13,180°)
Robber & Guard Rays	Robbers, Guards	(19,180°)

Table 4.4: The Ray Sensors and their features used during training. Each Ray Sensor can detect specific objects. A Ray Sensor has a number of rays and a total angle measured between its first and last rays.

Eleni Evripidou

Generic Observations	Dimensions
The agent's relative velocity	2
The room number the agent is currently placed in	1
Whether the three gates are open or closed	3
Whether the three gates are permanently closed	3
The relative coordinates of the three gates	6
Whether the alarm in each room is triggered	4
The number of collected valuables	1
Whether the agent has reached the goal	1
Whether the four valuables have been collected	4
The relative coordinates of the four valuables	8
The relative coordinates of the goal region	2

Observations of objects placed in the same room as the agent	Dimensions
Whether the two security cameras are on	2
Whether the agent is detected by the two security cameras	2
The relative coordinates of the two security cameras	4
The FOV parameters of the two security cameras	4
The relative coordinates of the guard	2
The relative velocity of the guard	2
Whether the agent is detected by the guard	1
The FOV parameters of the guard	2

Observations of fellow robbers	Dimensions
The relative coordinates of the fellow robber	2
The relative velocity of the fellow robber	2
Whether the fellow robber has reached the goal	1
The room number the fellow robber is currently placed in	1
The skills of the fellow robber	1

Extra Observations of Locksmith Agent	Dimensions
Whether they are able to open the three gates	3

Extra Observations of Technician Agent	Dimensions
Whether they can disable the two security cameras that are placed in the same room as them	2

Table 4.5: The common observations of robber agents are shown in the first three tables. However, each skilled robber observes some extra information about the environment described in the last two tables. The total number of data that the Locksmith agent observes is 64 and that of the Technician agent is 63.

Eleni Evripidou

is behind object B with height h_B and $h_A > h_B$, the agent shooting a ray at a height $h < h_B$ will only detect object B. This is desirable for the Front Rays since the agents should not be able to see through the furniture or walls.

On the other hand, it is impossible to include the detection of valuables in the Front Rays since valuables are placed in a higher position. The Valuable Rays are placed in a higher 2D plane compared to the other Ray Sensors and they can only detect valuables. Interestingly, these rays can intersect objects that belong to "Layer 1" like walls and gates but they cannot detect them due to identification reasons. So the agents cannot see valuables that are placed in different rooms. Security Camera Rays work similarly to the Valuable Rays. However, Robber & Guard Rays are different in the aspect that they do not intersect with walls, gates and furniture. This allows robbers to see guards and their teammates despite any obstacles in their way. For each type of Ray Sensor, the total observations are $N_R \times (N_D + 2)$ where N_R is the number of rays and N_D is the number of detectable objects. More specifically, we have a total N_D binary variables that represent whether or not the detectable objects were hit. In addition, we have one binary variable that can be true if the ray hits an object with a tag and false otherwise. The last parameter returns the normalised distance to the hit object, returning 1.0 when the hit occurred at maximum length and 0.0 when at length 0.0. Hence, we have a total of $(3 + 19) \times (4 + 2) + (15 + 2) + (13 + 2) + 2 \times (19 + 2) = 416$ observations from the Ray Sensors. Furthermore, Unity ML-Agents allows for stacking of Sensors. This allows the network to observe information than occurred in previous timesteps. In our case, we chose to stack the observations of the Ray Sensors, such that the agents remember objects they had seen for a short period of time. This is done to simulate the memory of a human player playing an FP game. We created 3 stacked sets of observations for each type of Ray Sensor, making the total observation size equal to $3 \times 416 = 1248$. This means that the agent observes the information Ray Sensors provide at timesteps: $t - 2$, $t - 1$ and t. Any stacked set of observations are stored in a circular buffer and this is fed to the neural network. The developers of ML-Agents Toolkit have specified that the stacked sets of observation are not the same as creating

Eleni Evripidou

a Recurrent Neural Network instead, n total number of stacked observations are fed to a simple Feed-Forward Neural Network.

The Vector Sensor is different for each type of robber, but they both observe some information about the environment including objects, the guard in the room they are placed in, and their fellow robber. These are described in detail on table 4.5. The extra observations for the Locksmith Agent are whether they are able to open the three gates and for the Technician Agent, whether they can disable the two security cameras that are placed in the same room as them . The resulting sizes of the Vector Sensors are 64 and 63 respectively.

We decided to help the agent by including the direct positions of valuables, gates and goal region despite that these are also included in the Ray Sensors. In our initial training sessions we did not include these parameters in the Vector sensor which resulted in the agent having difficulties spotting these objects.

The relative vector $\vec{u_{rel}} = \vec{u}|_S$ is defined with respect to the agent's local coordinate system S, whereas vector $\vec{u} = \vec{u}|_W$ is defined with respect to the global coordinate system W. The relative vector is also normalised with respect to the environment. So when the agent is placed on the far left side of the museum, and an object is positioned on the far right side of the museum, the distance would be 1.

The FOV parameters of an object $(|\vec{d}|, \cos\phi)$ are the distance $|\vec{d}| = |\vec{a} - \vec{o}|$, normalised with respect to the environment, between the agent's position $\vec{a}$ and that of the desired object $\vec{o}$, and the dot product $\cos\phi = \vec{f_o} \cdot \hat{d}$ between the forward direction of the object $\vec{f_o}$ and the normalised distance direction $\hat{d}$. The parameters are normalised such that $\cos\phi \in [-1, 1]$ and $|\vec{d}| \in [0, 1]$. Note that all vectors are in 2D space since there is no movement in the normal direction to the plane.

Eleni Evripidou

4.2.4 Rewards

Our reward function R_j includes sparse rewards and dense rewards, where $j = 1, 2$ indicates the reward function chosen. Sparse rewards are rewards given in rare events and dense rewards are given in multiple timesteps. Some reward signals are given during episode k at a timestep t, or at the end of episode k at $t_i = T_E \leq T$. The cumulative reward for an episode k is $R_{j,k} = \sum_{t=0}^{T} \gamma^t r_{t,j,k}$. In addition, our reward signals are either group or individual reward signals. Group reward signals are reward signals given to all members of the team simultaneously when an individual action has taken place. On the other hand, individual reward signals are received only by the agent who took a specific action. Group reward signals were used to induce teamwork.

We have been fine tuning the reward signals during different trials of training. We had to consider possible scenarios where robbers would prefer the option to give up and do nothing, rather than make the effort to steal and get caught. Furthermore, an important issue is the characteristic of selfishness in agents. Since our goal is to have a team of individuals that help each other, we had to find a way to reward this behaviour without giving too much direction. Our final results described in section 5 include training experiments using different rewards. The first reward function R_1 is described in table 4.7 and the second reward function R_2 is described in table 4.6.

The reward signal was tuned using a trial and error process. The reward functions R_1 and R_2 are very similar but have different ranges, specifically $R_1 \in [-4, 4]$ and $R_2 \in [-6, 3]$. We constructed R_2 to enforce stricter conditions when robbers lose. When using this reward function we used the changes in the environment explained in section 4.1.3. We removed the bonus reward $\frac{T-t}{T}$ to not encourage agents to finish early while being clumsy along the way. In addition, we decreased the value of the punishment for not entering the museum and changed it to an individual reward. Furthermore, we increased the weight of the punishment when the agents are detected by a guard, since we decided it too lenient previously. Lastly, we increased the weight w of the individual punishment for giving up wdt_{GU} because we have noticed that its

Eleni Evripidou

Group Reward $r_{1,k,t}$	Event	Type	Time
$+0.375$	collect a valuable	Sparse	t
$\frac{T-t}{T}$	all members have reached the goal region	Sparse	T_E
-2	at least one member of the team did not enter the museum	Sparse	T_E
-1	alarm is triggered at t	Sparse	$T_E = t + T_A$
$\frac{-0.5}{T_G}$	a guard detects an agent	Dense	t
$\frac{-0.5}{T_{SC}}$	a security camera detects an agent	Dense	t
Individual Reward $r_{1,k,t}$	**Event**	**Type**	
$+0.3$	agent exits the start region for the first time	Sparse	t
$+0.3$	agent enters the goal region or 2nd or 3rd or 4th room for the first time	Sparse	t
$\frac{-3dt_{GU}}{T}$	agent reaches the goal region at t_j and their teammate hasn't	Sparse	T_E

Table 4.6: The reward function $R_{1,k}$ for episode k, where t is the current timestep, T is the maximum number of timesteps, T_G is the maximum number of timesteps allowed in the guards' FOV, T_{SC} is the maximum number of timesteps allowed in the security cameras' FOV, T_A is the maximum number of timesteps allowed after the alarm is triggered, T_E indicates that the episode ended at time $t = T_E$ and $d_{GU} = T_E - t_j$.

value was very small if the episode ended early.

4.2.5 Training process

Architecture

The ML-Agents Toolkit constructs a fully connected neural network for observations that are in the form of Vector Sensors and Ray Sensors. The user can change the number of hidden neurons and layers by adjusting the hyperparameters of the policy. In our situation, we have two fully connected neural networks, one for the Locksmith

Eleni Evripidou

Group Reward $r_{2,k,t}$	Event	Type	Time
$+0.375$	collect a valuable	Sparse	t
-1	alarm is triggered at t	Sparse	$T_E = t + T_A$
$\frac{-1.5}{T_G}$	a guard detects an agent	Dense	t
$\frac{-0.5}{T_{SC}}$	a security camera detects an agent	Dense	t
Individual Reward $r_{2,k,t}$	**Event**	**Type**	
-3	the agent did not enter the museum	Sparse	T_E
$+0.3$	agent exits the start region for the first time	Sparse	t
$+0.3$	agent enters the goal region or 2nd or 3rd or 4th room for the first time	Sparse	t
$\frac{-3dt_{\mathrm{GU}}}{t}$	agent reaches the goal region at timestep t_j and their teammate hasn't	Sparse	T_E

Table 4.7: The reward function $R_{2,k}$ for episode k, where t is the current timestep, T is the maximum number of timesteps, T_G is the maximum number of timesteps allowed in the guards' FOV, T_{SC} is the maximum number of timesteps allowed in the security cameras' FOV, T_A is the maximum number of timesteps allowed after the alarm is triggered, T_E indicates that the episode ended at time $t = T_E$ and $d_{GU} = T_E - t_j$.

agent and one for the Technician agent. The Locksmith's NN has $1248 + 64 = 1312$ inputs, and the Technician's NN has $1248 + 63 = 1311$ inputs. Both NNs consist of 2 hidden layers with 256 neurons each. The output layer of both NNs consist of 9 neurons each, the same amount as the number of available actions.

During training we had 24 replicas in the Scene. Since we have eight possible museum map layouts, three of each layout were used as training environments.

We trained the agents with curriculum training and vanilla training to decide which is best. We discuss this process in the section 5.

Eleni Evripidou

Vanilla Training Type I

In this type of training we set the values of $T_G = 100$, $T_{SC} = 200$ and $T_A = 300$. The agents trained normally without any adjustment of variables along the process.

Vanilla Training Type II

In this type of training we set the values of $T_G = 100$, $T_{SC} = 200$ and $T_A = 10$. The agents trained normally without any adjustment of variables along the process.

Curriculum Training Type I

In this type of training we changed the value of $T_G =$ and T_{SC} during the learning process. Consequently this had an affect on the rewards assigned to the agent at each timestep. We implemented this to help the agents achieve their ultimate goal, to steal the valuables and reach the goal area. Since they need to learn how to avoid the guards and security cameras along the way, we decreased the values to the desired ones as such $T_{SG} = \{600, 500, 400, 300, 200, 100\}$ and $T_C = \{600, 500, 400, 300, 200\}$.

Curriculum Training Type II

In this type of training we changed the value of T_A during the learning process. We tried this since in Curriculum Training Type I, T_A was set to a large number. This allowed agents to escape within a long time frame even when the alarm was triggered. We wanted to discourage this behaviour, so in this training process we gradually decreased T_A as such $T_A = \{300, 200, 100, 10\}$.

Eleni Evripidou

4.3 Knowledge Gained

4.3.1 Initial Experiments

In our initial experiments, we trained the Locksmith and the Technician agent separately, in order to achieve the optimal observations and rewards. This stage helped us to solve issues and bugs we encountered with the mechanics of the environment.

The museum map used was different to the one shown in section 4.1.3. The environment layout was simpler, specifically the museum was a small room that included one gate and the valuable in the centre. Two security cameras were spawned in a random position around the museum and the museum spawned in a random position with random orientation. Similarly, the agent was spawned outside the museum in a random position and with a random orientation. A goal region was also randomly spawned in the scene. In figure 4.5 we show at a random layout of the environment. The security cameras and the gate were added to the scene depending on the agent being tested. For example, when we trained the Technician the gate was always open, and when we trained the Locksmith, the security cameras were disabled. Finally, we did not included guards for this experiment.

Eleni Evripidou

4.3.2 Curriculum Learning

The rewards at this stage were different. We assigned a reward equal to -1 when the agent reached the goal region without the valuable in their possession and $+1$ when the agent reached the goal region with the valuable. As long as the agent was alive we assigned $\frac{-1}{T}$ for each timestep, in order to discourage the agents from doing nothing and just wondering around. Due to the nature of the rewards, we noticed that, the agents learned to avoid the goal area, since at the first stages of training they did not have the valuable in their possession.

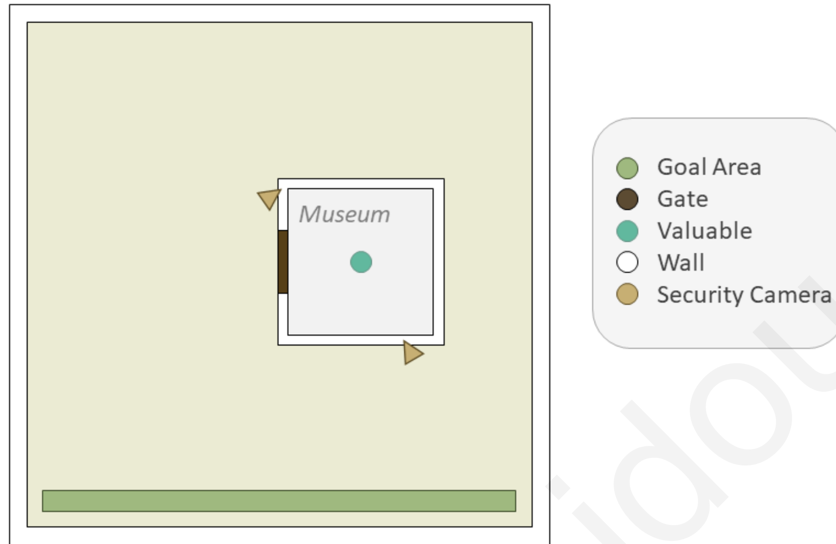

Figure 4.5: Layout of environment in the initial experiments.

Therefore, we added a curiosity module to the training to encourage the agent to explore more states of the environment. However, the results were not improved. Hence, we introduced curriculum learning. The first stage of curriculum learning was to initialise the agents with the valuable in hand, for them to learn that they should try to reach the goal area. The next stage was to place the valuable in a random position, such that the agents learn to grab the valuable before going back to the goal region. The third step was to introduce the museum. At this stage the gate was always open and no security cameras were involved. The last stage when training the Locksmith agent was to close the gate. On the other hand, during the training of the Technician agent, in the forth stage we introduced one security camera placed above the gate, and in the fifth stage we added a second security camera.

The results were promising in the case of the Locksmith agent. The agent learned to use their skills to open the gate, collect the valuable and go to the goal area. However, in the case of the Technician agent we noticed some issues with their behaviour. Although, the Technician learned to use their skills to disable the security cameras, they did not try to avoid the FOV of the security cameras. Hence, the behaviour they developed

Eleni Evripidou

was to steal the valuable and reach the goal region but being detected by the security cameras along the process and sometimes disable the security camera.

4.3.3 Group Training

Our next move was to train the agents together. In the first trial we ended an episode as soon as one member of the team reached the goal area. If any member of the team had already collected the valuable then the team won, otherwise they would lose. We trained the agents using curriculum learning, similarly to what we have explained above. The results were interesting since the Locksmith agent would collect the valuable and the Technician would stand outside waiting very close to the goal region. As soon as the Locksmith collected the valuable the Technician went to the goal region. However, the Technician did not use their skills to disable the cameras, and only waited for the Locksmith to complete the goals. This behaviour was enforced by the forth stage of the curriculum learning, since no security cameras were included and the gate was closed. Hence, only the Locksmith was able to open the gate and have access to the valuable.

Due to these issues, we decided to change the circumstances for ending an episode. We tried to end an episode only when all the members of the team have reached the goal. If any member of the team had collected the valuable successfully then the team would win, otherwise they would lose. However, the resuls were still not satisfactory, with one of the agents wondering around and not trying to steal and the other reaching the goal. So we decided to move on to the more complex and interesting environment described in section 4.1.3, since we had spent a lot of time in these trials that did not contain guards.

Eleni Evripidou

4.3.4 Setbacks

We have noticed some issues with the mechanics of the Ray Sensors during our experiments. When the agent was too close to a collider, for example a wall, its rays would pass through the wall. This meant that the agent would think that there is no obstacle in front of them. Hence, the agents would try to pass through obstacles and fail. To avoid this, we added a second, smaller collider inside the obstacles, so that when the agent got too close to them, the rays would intersect with the inside collider. This worked perfectly, and the agents did not get stuck on obstacles anymore.

The results were promising in the aspect that the agents learned to use their skills without receiving a direct reward for doing so. They also learned to collect the valuable and go to the goal region. However, we had noticed that the agents favoured a specific directional path. We assumed that this was due to the randomness of the museum position and orientation as well as the agent's. Some training trials produced a clockwise path and others an anti-clockwise path.

Another main issue we had faced is the observation of where the camera is directed towards, so that the agent would realise whether they are being detected by the security camera. Additionally, this would help the Technician agent find the direction from which it is safe to disable the camera. Since we provided all observation about position, velocities and directions of objects relative to the agent, we tried to do that for the direction of the camera as well. However, the independence of the horizontal/vertical movement of the agent and its orientation caused issues. As we have explained in section 4.2.2, the orientation of the agent is only changing due to the rotation action, so the velocity of the agent does not affect the orientation. For example, it is similar to a person walking in reverse without turning their head. So when the agent was moving towards the FOV mesh of the security camera the observation was the same as when moving away from it. To provide the agent with a meaningful observation about the direction of the security camera, we tried different different observation data. In the end, we decided on the FOV parameters $(|\vec{d}|, \cos\phi)$ we have described in section 4.2.3.

Eleni Evripidou

These parameters are independent of the orientation of the agent, and only depend on the agent's position, so the value is independent on the direction of the velocity of the agent.

Eleni Evripidou

Chapter 5

Results

Eleni Evripidou

5.1 Experiment I

In this experiment, we used Environment Type I and the reward function R_1. We train the agents with two methods, Vanilla Training Type I and Curriculum Training Type I for 24 hours each, completing 19×10^6 and 19.57×10^6 episodes respectively (see figure 5.1).

Both training types lead to similar results. The tactic of the robbers was to use their skills and to steal as many valuables as they could as quickly as possible, without being careful. Hence, they were seen by security cameras and guards, so the alarm was triggered. We noticed that this attitude was reinforced by the bonus rewards given to them for finishing their task early and by the value of $T_A = 300$, which was high enough for the agents to reach the goal before this time passed. In this case the robbers would win, since they would not receive the -1 punishment from the alarm. Nevertheless, the positive side is that the agents learned stay in close proximity during the episode, so we discovered we could force the agents to be more careful by tweaking some parameters.

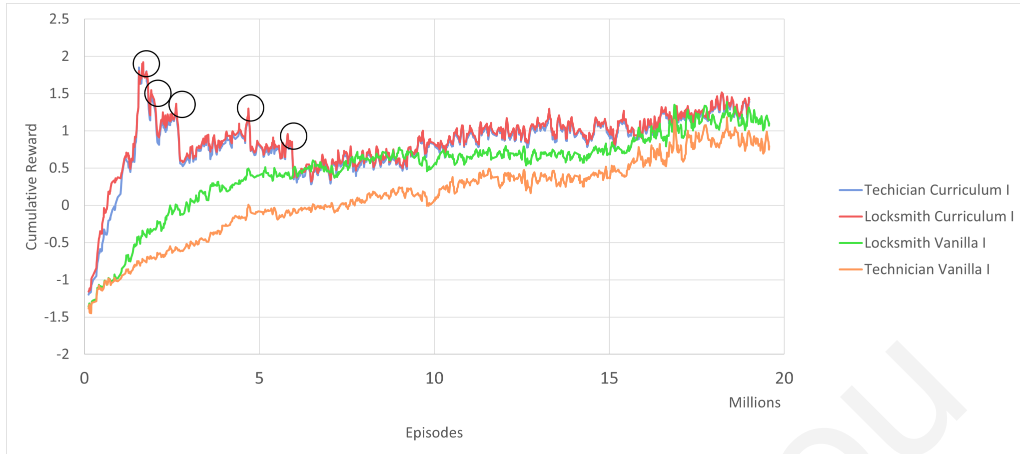

Figure 5.1: Cumulative reward received by Locksmith and Technician agents during Vanilla Type I and Curriculum Type I trainings with respect to episodes. We indicate in circles the start of new phases during the curriculum training.

On the other hand when comparing the two methods, in Curriculum Training Type I the Technician and Locksmith learned to stay together from early on. This can be seen by the figure 5.1. The cumulative rewards for both agents started to coincide following the completion of the first lesson. The higher values of T_{SC} and T_G were important since agents did not lose easily. Therefore, the agents learned that to steal valuables and reach the goal together was leading to higher rewards.

Alternatively, in Vanilla Training I the Technician was receiving a lower reward compared to the Locksmith. This was because the Technician took longer to learn to move through the rooms. So they were trapped in a room, trying their best to avoid the guards and security cameras. Consequently, the Locksmith learned to reach the goal before realising that waiting for the Technician to pass through the gate with them is more beneficial. This situation caused the team of robbers to lose very often, with the Locksmith being trapped in the goal region, and the Technician in a room.

However, these behaviours were affected by how the reward function R_1 was constructed. These results helped us to discover the issues with our environment setup and adjust the reward signals to generate more intriguing behaviours.

Eleni Evripidou

5.2 Experiment II

In this experiment, we used Environment Type II, the reward function R_2. We train the agents with two methods, Vanilla Training Type II and Curriculum Training Type II for 27 hours and 48 hours each, completing 22.4×10^6 and 44×10^6 episodes respectively. We chose to stop Vanilla Training Type II early since it was slower than Curriculum Training Type II (see figure 5.2). Although the agents seemed to be stacked for 5×10^6 episodes in a circumstance where they were getting very low cumulative rewards they finally overcome this. We can understand from the figure that at the start of the training, both agents exited the start region but lose very quick. At one point, one of the agents, specifically the Technician started being afraid to exit the start region, despite that the Locksmith did otherwise. Finally, the Technician overcame their fear so both exited the start region. We stopped the training process at the point where the agents find their way to the third room but they are detected by the guards or security cameras so the alarm is triggered. The agents develop similar tactics with Curriculum Training Type II method, but the training time is quite slow. This is due to the fact that the parameter T_A is set to a very small value, specifically at $T_A = 10$. Hence, the robbers lose immediately after the alarm is triggered. On the other hand, in Curriculum Training Type II, they have more time to find their way to the goal area, and learn their goal faster. When this parameter is decreasing gradually, the agents learn very quickly to adjust. So we decided to continue the training with the Curriculum Training Type II.

As opposed to the results form Experiment I, agents developed a more careful strategy such that guards would not detect them. The agents were also careful with respect to their exposure to the security cameras, but not as careful as with the guards. The first curriculum learning lesson ended at 33.39×10^6 episodes and was the longest one. The second lesson ended at 33.99×10^6 episodes, the third lesson at 34.59×10^6 episodes and the forth lesson lasted until the training was complete (see figure 5.3). During the first lesson the agents took a long time to learn their goal, regardless of the fact that

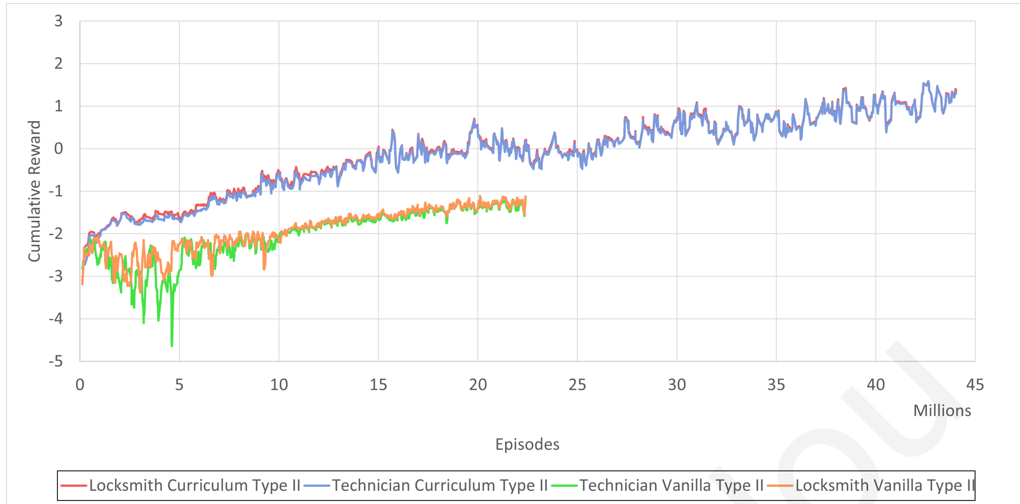

Figure 5.2: Cumulative reward received by Locksmith and Technician agents during Vanilla Type II and Curriculum Type II trainings. We indicate in circles the start of new phases during the curriculum training.

they had an additional $T_A = 300$ timesteps to win following the alarm trigger. In the following lessons agents learned their goal much faster, despite the fact that that T_A was decreasing down to $T_A = 10$.

The robbers are not always successful. There are many cases depending on the route that robbers should follow, where they often lose despite that we gave them enough time to win with how our parameters are set. It is worth mentioning, that most often robbers lose because of them being detected by the security cameras. So it might be a good idea in the future to increase the punishment for this event like we did for the guards.

However, this is acceptable since our goal was to understand how they can collaborate and what strategy they would develop and the results are quite promising. We have noticed that robbers try to stay close together as long as possible. It is a very interesting tactic and understandable. For example, there are circumstances when the Locksmith agent opens the gate, and the Technician agent does not pass through, but the Locksmith does. Now, the Locksmith agent has to find their way back to reopen the gate for the Technician to move on. This is dangerous since they are distracted from their main goal, which is to steal and not be detected by the guards and security cameras.

Eleni Evripidou

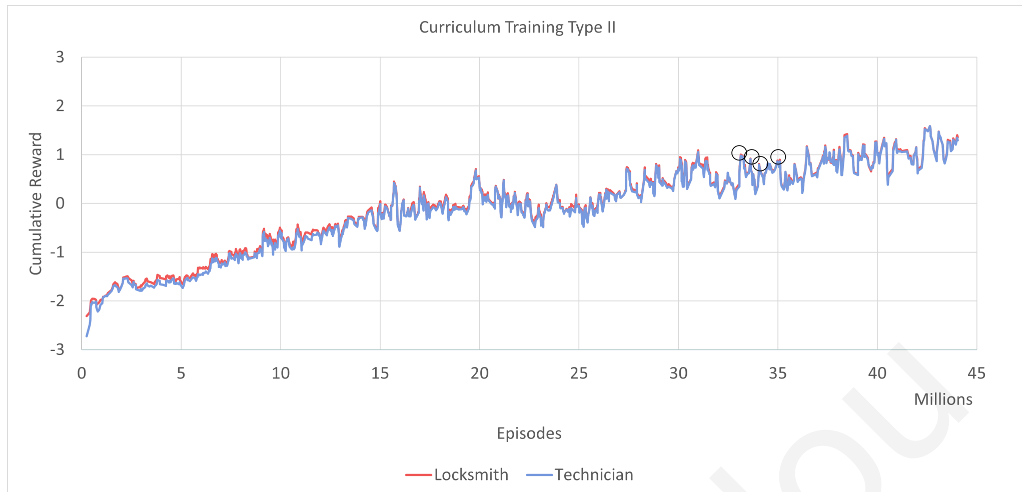

Figure 5.3: Cumulative reward received by Locksmith and Technician agents during Curriculum Type II training.

Now the Locksmith can be seen by security cameras that cannot be disabled by the Technician who is in a different room. On the other hand, the Technician is trapped in a room while trying their best to stay undetected. Hence, this strategy is quite smart. Furthermore, the robbers developed a tactic of moving in opposite direction to the guard that is in the same room as them. Each time the guard stops before changing direction, the robbers wait to "see" where the guard is headed, so that they move in opposite directions. This was very intriguing, since now the agents took their time to avoid the guards, rather than being quick as possible as they did in the Experiment I. Additionally, we have noticed that each robber looks in opposite directions while being together. One interpretation is that they try to communicate what they "see" from Ray Sensors, through each other's velocities. We have noticed that the Locksmith's orientation is such that they are looking for the guard of the room they are in, and the Technician's orientation is such that they are looking for the guard in the next room. A counterexample to this interpretation is that occasionally, the robbers are detected from a guard that passes in front of the security gate which is currently being opened by the Locksmith. In addition, the agents are more afraid of the guards compared to the security cameras. This was a result of our reward function, so in future experimentations we should increase the punishment of being inside the FOV of a security camera.

Eleni Evripidou

5.2.1 Environment Changes

We use the results of Experiment II training while changing some of the parameters of the environment. For example, we decrease the time T_{SC} and T_G so that the robbers lose quicker. These two parameters are not observable by the robbers. The agents seem to be okay with those changes, and keep their tactics as they are but they lose more often due to them being in the FOV of the security cameras. We figured out by giving this parameters as an observations accompanied with the current time period dt that they have been detected by guards and security cameras, would help agents to adjust their behaviour accordingly. In addition, we increase the velocity of guards, which did not affect much the agents performance. The thing that affected the most, is changing the guard routes such that at the start of an episode the guard passes through the start region. The robbers seem to find a way to not be detected if the guard is moving from their right to the their left because this case was included in one of the training environments. However, if the guard is moving from their left to their right the guards, the agents assume that any guard standing to their left would move away from them. So the agents enter the museum quickly and do not wait to see where the guard is headed. This behaviour is affected by the predetermined routes we introduced. Nevertheless, if we design routes such as all cases are included, agent will learn to wait guards to start moving before entering the museum, so they can detect the guard's velocity.

Eleni Evripidou

Chapter 6

Discussion

6.1 Limitations

Our results are promising, however they come with some constraints. We choose to use a different policy for each skilled agent, which introduces undesirable obstacles that could have been avoided. For example,both agents must learn to navigate, independent of their skill. Each policy has to find actions to help agents navigate from scratch. This introduces an unnecessary time consumption in the learning process.

In addition, the resulted behaviour is not comparable with that of humans. The current agents would probably lose while playing against humans, if not always, very often. This is a result of how we coded the behaviour of guards and security guards, giving the robbers enough time to find a way to win. It would be interesting to see how their strategy is compared while training against guards that learn along with them using self-play methods.

Furthermore, some parameter changes in our environment, such as the routes of guards, affect the behaviour of agents. The team of robbers lose very quick due to this, however, we could improve this behaviour by including all types of routes in our training process.

Eleni Evripidou

6.2 Future Work

Taking the limitations of our work into account, we realise that training one policy for each team might improve our results. So in that case, each agent in a team will share the same brain. We can add their skill as an observation. This limit us to give the same observations despite the current skill of each agent. For example, if we use the same observations, as in this project, we have to construct an agent which takes 1315 observations. The number of actions will also have to include all possible actions that each skilled agent can do. So in our case we will have to include two continuous actions and four discrete branches. Using a flag we can allow the appropriate action depending on the agent's skill to take place. Hence, let's say that an agent controlling a Locksmith robber, increases the probability to disable a security camera, this would not affect the environment and would not maximise their reward. Hence, the policy will decrease the probability of this specific action happening in the future when the controlling skill is that of the Locksmith. Choosing to train one policy will reduce the required time for training the agents to navigate, which is a limitation of our current strategy. We assume that training one policy will also help the team as a whole. Since agents observe the skills of their fellow robbers anyway, they might learn what their teammates abilities are faster and use them in their advantage.

Furthermore, it might useful to disentangle all the Ray Sensors so that each can detect one object type. This introduces a higher size of observations but can lead to the agents learning the environment faster. We have already seen this have a positive effect when we disentangled the detection of robbers and guards from all the other objects in Layer 1.

In addition, we desire to train the team of guards alongside the team of robbers. We can do this using self-play, which is provided by ML-Agents. This will be easier to implement if we choose to use one policy for each team. The Unity ML-Agents Toolkit has already established a new algorithm called MA-POCA which assigns group

Eleni Evripidou

rewards and deactivates agents, while they are still receiving rewards, during an episode. [17]. We implemented our own version of this in this project as discussed in section 4.2.1. We unfortunately could not use the ML-Agents' implementation since MA-POCA algorithm can only work with a team of agents that share the same policy.

If the above ideas work, we can also consider to more ambitious concepts. An initial goal for this project was to introduce a variable that adds an aggressiveness or reluctance trait in the behaviour of the agent. Similar to how human players behave when controlling a character in multiplayer games. Each one has their own strategy, some are more aggressive, some are more careful etc. It would also be interesting for a team of agents to decide which and how many skilled characters to choose depending on the museum layout, the security cameras, the number of rooms, gates and valuables.

Eleni Evripidou

Chapter 7

Conclusion

The work done in this project is the first step in understanding how a team of agents with different skills can compete against an opponent team with a different role, while developing strategies that use their skills to assist their team rather than just themselves without awarding them directly for it.

We have encountered multiple obstacles while trying to achieve this. We have overcome some of them, and finally produced some satisfactory results while training each agent using a different policy. The team of robbers containing two agents with different skills managed to cooperate and generate a strategy to avoid guards for as long as possible. We realised how important the construction of the reward function in defining the agents' strategy. In addition, we discovered that introducing curriculum learning methods in various ways depending on the environment, can assist in speeding up training. Although our results are interesting, changing some parameters of the environment affects the performance of the agents. This is a result of the training environment design. A more thorough environment design is required to produce behaviours that can adjust to such changes. Furthermore, we plan to include the opponent team in our training, by using self-play, in our future work. Nevertheless, to train each agent with a different policy increases the training time. This can be avoided by using one policy. In the

Eleni Evripidou

future, we plan to experiment with this option, to compare how much faster the agents learn and if the resulted strategy is comparable with our own.

Although our results do not surpass human performance, they can produce interesting behaviours for NPCs in video games. Developers in game industries could use this method to generate more intriguing strategies for NPCs, where they cooperate to defend against human players.

Eleni Evripidou

Bibliography

[1] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.

[2] "Deep blue." (1997), [Online]. Available: `https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/`.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[6] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv: 1712.01815*, 2017.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[8] C. Berner, G. Brockman, B. Chan, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[9] B. Baker, I. Kanitscheider, T. Markov, *et al.*, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

Eleni Evripidou

[10] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[12] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[13] "Unity documentation, version 2020.3, unity manual." (2022), [Online]. Available: `https://docs.unity3d.com/2020.3/Documentation/Manual/index.html`.

[14] A. Juliani, V.-P. Berges, E. Teng, *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020.

[15] "Unity ml-agents toolkit github." (2022), [Online]. Available: `https://github.com/Unity-Technologies/ml-agents`.

[16] "Agent learns to jump a wall." (2022), [Online]. Available: `https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-Curriculum-Learning.md`.

[17] A. Cohen, E. Teng, V.-P. Berges, *et al.*, "On the use and misuse of abosrbing states in multi-agent reinforcement learning," *RL in Games Workshop AAAI 2022*, 2022.

Eleni Evripidou