AI-ORIENTED COMMUNICATION IN A GAME ENGINE WITH HYBRID MULTI-AGENT SYSTEMS USING REINFORCEMENT LEARNING

DOMINIC ARMAND STAMATOIU

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE IN COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE

DEPARTMENT OF
COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

STUDENT NUMBER

2051167

COMMITTEE

dr. Murat Kirtay
dr. Merel Jung

LOCATION

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science &
Artificial Intelligence
Tilburg, The Netherlands

DATE

June 23, 2023

WORD COUNT

7400

ACKNOWLEDGMENTS

I would like to acknowledge and give my warmest thanks to my supervisor, family and friends for the continuous support.

AI-ORIENTED COMMUNICATION IN A GAME ENGINE WITH HYBRID MULTI-AGENT SYSTEMS USING REINFORCEMENT LEARNING

DOMINIC ARMAND STAMATOIU

Abstract

This project aims to present the training of an AI agent in a virtual environment using reinforcement learning and to build a game that facilitates communication between AI and other non-intelligent agents. The research goal is for the AI agent to be able to differentiate between the non-intelligent agents and to finish the game with the maximum reward possible. The game engine used is Unity which contains a package that is able to introduce the reinforcement learning algorithm into the game environment. The game focuses on the task of the AI agent to go through an environment made out of tiles, each tile giving a different reward. The AI agent is able to train effectively, by bringing together the exploration of the environment as well as the information given by the non-intelligent agents through communication. After training the AI agent in three different environments, the conclusions are that it is able to achieve the maximum cumulative reward possible by choosing the best path in the environment as well as differentiating between the non-intelligent agents.

1 DATA SOURCE, ETHICS, CODE, AND TECHNOLOGY STATEMENT

The data and figures created for the purpose of this research were generated by the programming and training of the AI agent. Some coding parts have been adapted from a YouTube video and from the documentation of the ML-Agents package. The references for these coding adaptations can be found in the coding component assignment. Two different tools were used in order to check the grammar and spelling of the thesis text. One tool was Trinka while the other was Grammarly.

2 INTRODUCTION

AI advancements have reached video games as new AI agents are created and tested in game environments by using reinforcement learning and other tools designed to teach them new behaviors. The focus of this project is to display the communication between AI and other non-intelligent agents and the training of the AI agent in a game environment using reinforcement learning. The non-intelligent agents in this project are defined as agents that hold information about the environment and that respond to the requests of the AI agent, however, they cannot learn anything or come up with strategies during the game. Through the use of both non-intelligent and AI-oriented agents, the project focuses on a gathering of hybrid agents and the communication that results from their interactions. The game engine that is used is Unity and it includes a package called "ML-Agents" which supports the creation and training of AI agents. The AI agent communicates with three non-intelligent agents in order to get to the other side of a square area which is equally divided into tiles of the same size. Each tile has a different color: red, orange and green. Multiple colored tiles are present in the environment, and each color corresponds to a unique reward for the AI agent. Each of the three non-intelligent agents deployed is assigned to a specific color and possesses knowledge of its location.

Reinforcement learning is crucial when training the AI agent, as it helps it to learn by interacting with the environment. The AI agent makes an action, receives feedback in the form of rewards or penalties and then adjusts its strategy accordingly. By using deep reinforcement learning, the AI agent is also able to use neural networks to further enhance its training, by arriving at a more complex algorithm that is better at optimizing the learning process. The two types of agents present in this project are the AI-oriented agents and non-intelligent agents. The AI agents can be powered by algorithms such as reinforcement learning to improve their performance over time. On the other hand, the non-intelligent agents, follow pre-programmed rules and do not possess the ability to learn or adapt. They provide a deterministic response based on their programming and the input they receive.

In terms of actions, the AI agent is allowed to rotate to the left or right and it can also move forward and perform a dialogue towards the non-intelligent agents, such being required for gathering information. The non-intelligent agents are static; they stay in the same place during the progression of the game. The only action that they have to perform is to respond to the AI agent's questions. The AI agent perceives the environment by observing its own position and rotation as well as the

position of the goal. It also processes the information from the dialogues given by the three non-intelligent agents to better map the environment around it. When looking at the non-intelligent agents, they are only given the positions of their respective colored tiles at the beginning of the game. Through their communication with the AI agent, they are also provided with the location of the AI agent in order to better instruct it on how to move through the environment.

The environment in which the agents are situated can be interpreted as a battlefield, in which the agents are presented as tanks. Communication is important on the battlefield as it can "ensure successful outcomes" and it has to be as precise and optimal as possible (Mmisupport, 2023). By making a parallel between the project environment and the battlefield, it is also important which agent or soldier shares the best information or gives misleading information and can be considered an enemy that should be ignored.

In the articles published by Juliani et al. (2018) and Urmanov, Alimanova, and Nurkey (2019), the importance and ease of using Unity and the "ML-Agents" package in the building of simulations and environments for use in research is highlighted. In addition to that, the study by Cao and Lin (2019) and a later research paper by Cao, Wong, Bai, and Lin (2020) evidentiate the crucial use of non-intelligent agents and their much-needed help in teaching the AI agent about the environment.

The goal of the AI agent is to effectively differentiate the non-intelligent agents by communicating with them and choosing the most reliable one, in order to finish the game with the maximum reward possible. Meanwhile, multiple graphs made using TensorBoard will aim to show the evolution over time of the AI agent in its training and to motivate the behavior that is presented by it. Multiple different-sized environments will be available for the AI agent in order to draw conclusions regarding the behavior change.

The main findings of this research are that the AI agent is able to achieve the maximum reward possible by participating in dialogues with the other non-intelligent agents in order to get information about the environment. In addition, the AI agent is able to pinpoint the agent with the most reliable information, proving that it can differentiate the non-intelligent agents. This project aimed to add knowledge to the existing literature about developing communication between different types of agents in a virtual environment and also about training an agent with reinforcement learning to acquire an agent differentiation skill.

3 RELATED WORK

This project focuses on building the virtual environment, training the AI agent, and presenting the communication between the agents. The virtual environment that will be used is Unity, which provides support for making the game accessible and visually pleasing. According to Juliani et al. (2018), present-day game engines like Unity, "are powerful tools for the simulation of visually realistic worlds with sophisticated physics and complex interactions between agents with varying capacities". Among the toolkits and packages that Unity presents to its users, the Unity Machine Learning Agents Toolkit (ML-Agents) is the most important one for AI researchers as it brings together the Unity engine and reinforcement learning (Urmanov et al., 2019). By using the ML-Agents Toolkit, the agent will be allowed to make mistakes repeatedly so that it learns to perceive the difference between the most rewarding actions and the ones that lead to a significantly lower reward (Baby & Goswami, 2019). Apart from being used for training, the ML-Agents package works with TensorBoard, "which enables the continuous observation of statistics of the model" (Lukas, Tomičić, & Bernik, 2022). TensorBoard provides tables and figures depicting the data collected during the training of the AI agent, some examples of tables being: the cumulative reward and episode length, both presented over time.

As stated previously, the AI agent will be trained by using reinforcement learning, which is defined by Botvinick et al. (2019), as being a machine learning algorithm that "centers on the problem of learning a behavioral policy, a mapping from states or situations to actions, which maximizes cumulative long-term reward". Reinforcement learning is known for applications in machine automation, self-driving cars, and finance, but this project focuses on its implementation in robotics and gaming (Mwiti, 2023). The AI agent is required to maximize its reward by training in the Unity environment. Reinforcement learning presents a repeated process in which the AI agent chooses an action in the environment through which the next reward and state of the agent is decided.

Deep reinforcement learning is considered a modern reinforcement learning algorithm that also relies on neural networks. Neural networks are an important addition because they can handle high-dimensional data and can learn complex non-linear relationships between states and actions. One technique that is cataloged as a deep reinforcement algorithm is the proximal policy optimization (PPO). According to Wang, He, and Tan (2020), PPO "is a deep reinforcement learning algorithm, that performs well without extensive hyperparameter tuning", is easy to implement, and is also efficient in solving complex tasks (Vanvuchelen, Gijsbrechts, &

Pardalos, 2020). Because of these advantages, PPO is considered one of the best reinforcement learning methods and it was included as the most commonly used algorithm in Unity when the ML-Agents package is added (Cao & Lin, 2019). Additionally, according to Youssef et al. (2019), adding imitation learning on top of reinforcement learning makes the AI agent to learn better and faster the task at hand. Imitation learning is a machine learning algorithm where the agent learns to perform a task by imitating the behavior of a human. Adding imitation learning could be a future direction of enhancing this project, as it could boost the performance of the agent, but the use of imitation learning is not the focus of this thesis.

In his research, Andersson (2021) argues that changing the hyperparameters used when training, dramatically influences the behavior or results of the trained agent. While for the small and simple environments, the default hyperparameters can be used, for the more complex and big environments, it is necessary to test different configurations of hyperparameters, which could bring out the necessary results that are needed from the agent. It is also noted that the results can be improved if the hyperparameters known as the batch size, buffer size, and number of layers are increased (Andersson, 2021).

Apart from PPO, there are also other high-performance reinforcement learning algorithms presented in research papers, such as Deep Q Networks (DQN). The main difference between PPO and DQN is that the former is an on-policy algorithm that learns from the most recent policy's data, while the latter is an off-policy algorithm that can learn from older, stored data. The AI agent presented in this thesis is more compatible with PPO, as the algorithm is more stable and uses a mechanism to prevent the policy from changing too much in a single update. According to Mehta (2020), Evolution strategies (ES) is a state-of-the-art reinforcement learning algorithm as it "is much more efficient and faster than other RL algorithms with the only drawback that the data used for its training acquires a lot of memory". It is a unique reinforcement learning algorithm as it focuses on the evolution of the agent by training its offsprings in parallel from one generation to the other in order to find the best descendant for a task at hand (Mehta, 2020). In the study by Patacchiola and Cangelosi (2016), Bayesian Networks (BNs) are used as part of the reinforcement learning framework by helping to differentiate between agents based on trust and belief when looking at the reliability of the information given by them. It was acknowledged that "BNs have the potential for being included as trust evaluator modules in robotic systems" (Patacchiola & Cangelosi, 2016). PPO is a better choice as the reinforcement learning algorithm for this project because ES is too advanced and requires a lot of memory space, while BNs rely heavily on probabilities and dealing with uncertainty.

Communication between the agents will be present in the virtual environment in the form of simple dialogues focused on transmitting information about the environment to the AI agent. The AI agent will ask one of the non-AI agents for help, and if the tile that the non-AI agent represents by color is next to the AI agent, it will give the AI agent the action that is necessary to be taken in order to arrive on that specific tile. The content of the dialogues between the agents will depend on the tiles that are next to the AI agent and which of the non-AI agents it wants to communicate with. According to Dorri, Kanhere, and Jurdak (2018), one of the most used approaches in the communication of agents is the speech act, which states that agents communicate verbs and sentences that relate to changes in the environment around them and also that "an agent can act as a speaker (S) that produces utterance to change the beliefs of the hearer (H)".

Focusing on the importance of the non-intelligent agents, the study by Cao and Lin (2019), confirms that using multiple agents that observe the actions of the main agent, which has to go through the environment, is beneficial as it can increase the speed of the training. In a follow-up research paper, Cao et al. (2020) argue that there is a noticeable difference in training using Unity, between hierarchical and non-hierarchical reinforcement learning. Hierarchical reinforcement learning is used in the mentioned paper by setting an agent that can observe the environment globally and which gives information to the agents that are moving in the environment about how to achieve or arrive at the goal (Cao et al., 2020). This property is also available in the making of this project in which agents that take the role of spectators, guide the AI agent through the environment and overall decrease the training time needed for the development of the desired behavior.

There are also articles that focus on the communication between agents, but also on the differentiation of agents. In the first article by Kirtay, Oztop, Kuhlen, Asada, and Hafner (2022), it is presented a Nao robot that is tasked to differentiate between heterogeneous agents, meaning both robot and human agents, by making a "trust" bond with the most reliable agents. The Nao robot, which is trained using the SARSA reinforcement learning algorithm, can properly differentiate the agents as it chooses the ones "with reliable guiding strategies" in order to complete the task at hand (Kirtay et al., 2022). This article further encourages the idea that the AI agent can be trained to differentiate between other agents by using reinforcement learning, however, a difference relies on the fact that it is not known if the differentiation will also work in a virtual environment like Unity. A second article by Sukhbaatar, Szlam, and Fergus (2016), shows the communication between non-heterogeneous agents, which are similar agents, by using deep learning architectures instead of reinforcement

learning. It is demonstrated that the agents arrive at better results when communication is present than without it (Sukhbaatar et al., 2016). In addition, a new functionality is added making "possible to interpret the language devised by the agents, revealing simple but effective strategies for solving the task at hand" (Sukhbaatar et al., 2016). Overall, it is presented that communication between agents can bring better results and could help the AI agent in this project to arrive at better cumulative rewards, a difference being that the AI and non-intelligent agents are part of a heterogeneous group. This project can help bridging the gap between communication in a virtual environment and communication applied to a gathering of hybrid agents.

4 METHOD

The software used is the game engine Unity with its Machine Learning Agents Toolkit (Unity-Technologies, n.d.). As explained previously, Unity offers the opportunity of creating sophisticated simulations and games, while the ML-Agents Toolkit brings in the reinforcement learning aspect, that enables the user to train agents. The programming languages used are C# and Python. C# is used in Unity to define the agent's behavior and interactions by coding scripts and modifying the agent's parameters. In this project, there is no Python code provided. However, the Python language is involved indirectly, as the agent's framework uses Python for setting up and running the training process, and for communicating with the Unity environment during training.

Python's API (Application Programming Interface) is an essential part of the Unity ML-Agents framework as it brings together the Unity environment, where the agent's behavior and interactions are defined using C# scripts, and the machine learning algorithms implemented in Python using TensorFlow or PyTorch, which are useful for training the agents. For the Unity Project, version 2021.3.11f1 is used, while Python uses version 3.9.13. The reason behind the use of specific versions of software is that the ML-Agents Toolkit is compatible with only a few versions. Packages are installed in the Unity environment, the most important one being the "ML Agents" package. Additional packages include: "TextMeshPro" for customization of on-screen text, "Mathematics" for the use of vectors, and "Visual Studio Code Editor" for enabling developers to use Visual Studio as their primary code editor when working on Unity projects. The training of the AI agent is done with help from the GPU, which is generally better than training with a CPU for reinforcement learning tasks in Unity because it can greatly reduce the training time and helps to iterate and experiment more quickly compared to using a CPU.

The machine learning algorithm used is reinforcement learning because the AI agent has to learn how to perceive the environment based on the rewards that appear when interacting with it. This type of learning is applied only to the AI agent, as the other agents are not AI-oriented. The reinforcement learning technique used is PPO as it is efficient at solving complex tasks and is included in the ML-Agents package. Considering that the training of the AI agent takes between 30 and 130 minutes and that the AI agent needs to develop a specific behavior based on the rewards and hyperparameters provided, PPO is a great choice of an algorithm as it is not extensive in hyperparameter tuning.

The results are evaluated using the graphs made by TensorBoard, which enable us to more easily interpret the data. An example could be a reward graph that shows the AI agent's cumulative reward over time. This can be useful for understanding how the AI agent's performance improves as it learns to perform the task. Another example would be a histogram of rewards, which shows the distribution of rewards received by the AI agent during training. This presents how the AI agent's performance varies across different episodes of training. The AI agent is also evaluated by looking at the number of actions that it takes on each episode. The AI agent should start with several actions taken as it explores the environment, but over time the number of actions should decrease as the AI agent moves closer to discovering the optimal route. Multiple different-sized environments will be available in order to see the differences in the behavior of the AI agent when being placed in much bigger and more complex environments than the previous ones.

The tiles have 4 colors: blue for the beginning and end tile, green for the highest reward tile, red for the lowest reward tile, and orange for the tile with the reward amount situated between the green and red tile. The reward for each tile is given only once per episode or iteration, as the AI agent would take advantage of the bad implementation and would only move to the green tile repeatedly without finishing the game. There is a positive end reward, a positive reward for stepping on the green colored tile, and a positive reward for when the AI agent displays a message for one of the other non-intelligent agents in order to encourage it to communicate with the other agents. There are also negative rewards with the purpose of punishing the AI agent: when stepping on the red tile, when the AI agent performs a rotation in order for the AI agent to make as few rotations as possible, when the AI agent goes outside of the designated environment comprised of tiles, and for every extra step that the AI agent takes in order to make it finish the game as quickly as possible. The reward given to the AI agent when moving to the orange tile is expected to have a value approaching zero.

Apart from there being positive or negative rewards, there are also high or low rewards, meaning that for example, an action containing a high positive reward should be strongly followed while an action containing a high negative reward should be avoided as much as possible. The action that is taken to arrive at the end tile of the game or at the barriers that are outside of the environment as a border yields a high reward as they are the actions that end the game or episode. On one hand, when the AI agent arrives at the end tile, he receives a high positive reward because he completes the main objective of the game. On the other hand, when the AI agent hits a barrier, he receives a high negative reward, as he is no longer in the environment where the game takes place. In both cases, the game ends and the AI agent is sent to the start tile for the next iteration of the game.

As stated previously, there are four agents in the environment. Three of the four agents are non-intelligent agents that provide the AI agent with information about the environment. All agents have the same texture of a tank as depicted in Figure 1, but they differ in color to signify the different roles they play in the game. Each non-intelligent agent has the color of the tiles they hold information about: green, red or orange, while to the AI agent is given the color brown.

Figure 1: The AI agent

The first step in creating the main environment, which has one AI agent and three non-intelligent agents, was the creation of a prototype with only two non-intelligent agents out of the three. This prototype was required in order to test the Unity environment as well as the training process of the AI agent. In Figure 2, you can see the prototype, the first environment built for this project. It is composed of an arrangement of two-by-two tiles. The blue tile on the left is the start tile, while the blue tile in the

bottom-right is the end tile. The AI agent begins the game on the start tile and is encouraged to go through the environment, by first stepping on the green tile as it yields a higher reward than the red tile and then on the end blue tile in order to complete the game. By stepping on the surrounding brown area, the AI agent would encounter the barrier and would be sent back to the start tile for the next iteration.

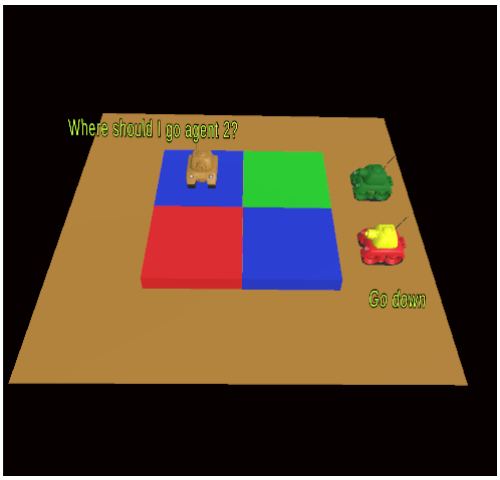

Figure 2: The prototype (two-by-two tiles environment)

In terms of actions, the AI agent has three discrete action branches between which it can shift. The action branches are discrete because they present discrete variables and not continuous variables that can be included in an interval between two values. The first action branch deals with rotation, so that the AI agent can choose between not rotating, rotating to the right by 90 degrees, or rotating to the left by 90 degrees. The second action branch is concerned with moving forward so that the AI agent can choose between not moving or making a step forward to another tile or barrier. The third action vector is concerned with the dialogue performed by the AI agent towards the non-intelligent agents. In the prototype, because there are only two non-intelligent agents present, the AI agent can choose between not displaying a message, addressing a message to the green agent, which is considered "agent 1" (as the AI agent is not made aware of the non-intelligent agent's role), or addressing a message to the red agent, which is known by the AI agent as "agent 2". In the other environments, the AI agent can also address a message to the orange agent known as "agent 3". In addition to the dialogue choices, the AI agent can perform a dialogue only after an action regarding movement is made and only once, so that the AI agent does not just keep asking questions and increasing its reward for performing dialogue, without ending the game. As you can see in Figure 2, there is an example of a conversation between

the AI agent and the red agent, in which the red agent, highlighted in yellow, indicates the closest red tile to the AI agent. The AI agent holds this type of information by communicating in the form of observations on which it can act or not.

The AI agent is also observing the environment and even analyzing itself. Firstly, the AI agent collects observations about its own position and rotation in the environment. It also knows the position of the goal which is the end tile. Other important observations are the incoming dialogues from the red, green, and orange agents, which provide critical information about the environment. The dialogues are transformed into integers and inform the AI agent of different recommendations coming from the non-intelligent agents about which actions should the AI agent take. There are five possible dialogues spoken by the non-intelligent agents: "No nearby tiles", "Go up", "Go down", "Go left" or "Go right". The dialogues of the non-AI agents indicate where the closest tile that they represent is. They share the information containing the direction and if the tile is to the left, right, down or up. The information is chosen from the perspective of the spectator agents, which have a bird's-eye view of the whole environment; therefore, if they tell the AI agent to go to the right, that actually means that the AI agent should go to his left side. If the AI agent is already on a tile and there are no other tiles with the same color around, the non-AI agent that deals with that color will tell it that there are no nearby tiles. All the observations taken by the AI agent are used by the neural network in order to make decisions about how should the AI agent act in the environment.

The main parameters that are required for the AI agent to develop the needed behavior are the rewards and hyperparameters of the PPO algorithm. The training is focused on finding the best rewards and hyperparameters for a given environment size. In the Unity environment, before starting to train an AI agent, it is crucial to create duplicates of the environment that you are working on, so that there are more AI agents in more environments that train at the same time. This strategy results in faster training of the AI agent and in an easier visualization of the strategies and behaviors that the AI agent can learn during training. Another useful visualization aspect that was added, was changing the color of the environment to green when the AI agent finished the game by stepping on the end tile. At the same time, if the AI agent went into a barrier by stepping out of the environment or if the game ended in any other way, the environment would turn red.

The first game environment that was created was a two-by-two environment made out of tiles as the prototype with only two non-intelligent agents. The two-by-two environment was duplicated, resulting in nine environments of the same size. Considering that the study by Andersson

(2021) specified that the hyperparameters known as the batch size and buffer size are important and should be increased the more complex the environment is, then for a simple environment like the prototype they should be decreased. According to Lukas et al. (2022), the "batch size is the number of experiences in each iteration of the gradient descent", while "the buffer size is the number of experiences that must be collected before updating the policy model". The default value for the batch size is 1,024, while for buffer size is 10,240, so a good strategy was to test these hyperparameters and make them smaller and smaller by dividing them by two. By trial and error, there appeared a training iteration where the AI agent achieves the required behavior and where all the hyperparameters are the default ones except for the batch size of 32 and buffer size of 512. The rewards used are in Figure 3, and they were not changed during the many iterations of training. An unfortunate behavior of the AI agent which would start in the early steps of the training process, was that it would pick rotating actions repeatedly on a tile and never finish an episode, thus not being able to train properly. A solution to this behavior was adding a maximum rotation counter of ten rotations when the AI agent sits on a tile. If the AI agent achieved the maximum rotations, it would get a high negative reward and the episode would end by putting the AI agent back on the start tile.

stepping on the red tile	-2
stepping on the green tile	2
stepping on the end tile	5
moving forward	-0.3
rotating	-0.3
2 rotations in the same direction	-0.4
dialogue action towards a spectator	0.1
going out of bounds/ into a barrier	-5
achieving the maximum rotations on a tile	-5
Maximum reward possible in an episode	6.2

Figure 3: Rewards for the two-by-two tiles environment

The second environment that was created was the three-by-three tiles environment depicted in Figure 4, in which the third spectator called the orange agent was added. Apart from being duplicated into 9 environments, the training time of the 9 AI agents was increased by changing the "max steps" hyperparameter to 2,000,000 as it was observed that more training resulted in better end rewards. A step, in this case, is an iteration of the reinforcement learning algorithm. The AI agent receives some observations and chooses an action to send to the environment, which responds by updating the AI agent's state and returning a new observation and

reward. At the end of the iteration, the AI agent uses the new reward and observation to update its policy or strategy. This is the main reason for increasing the steps that the AI agent will take during training, so that it can use more data for a better policy.

Since the article by Lukas et al. (2022) recommended increasing the distance between the batch size and buffer size, the batch size was systematically divided by 2 until the training ended with good results, the buffer size being equal to 4,096, while the batch size was equal to 64. The hyperparameter "num layers" was increased to give the network more computational power.

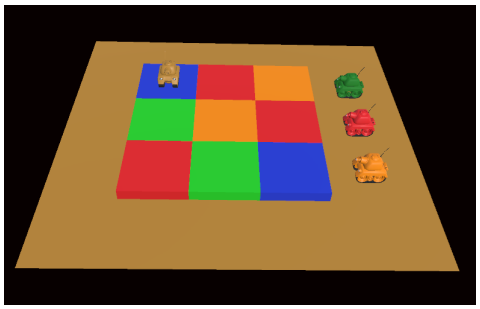

Figure 4: The three-by-three tiles environment

In Figure 5, it is shown that in terms of rewards, in the three-by-three tiles environment, the reward for the movement of the AI agent, meaning the forward movement and the rotation, were increased in order to allow the AI agent more freedom in exploring the environment, without overusing movement-related actions. Adding to that thought, the negative reward given to the AI agent for doing 2 rotations in the same direction was removed. Finally, considering that the orange agent was introduced, a new reward was made available for the AI agent when stepping on the orange tile.

stepping on the red tile	-2
stepping on the green tile	2
stepping on the orange tile	0.1
stepping on the end tile	5
moving forward	-0.2
rotating	-0.2
dialogue action towards a spectator	0.1
going out of bounds/ into a barrier	-5
achieving the maximum rotations on a tile	-5
Maximum reward possible in an episode	8.4

Figure 5: Rewards for the three-by-three tiles environment

The third and final environment that was created was the four-by-four tiles environment, which is the most complex and can be seen in Figure 6. The number of duplicated environments and of the max steps remained the same. Due to the rising complexity, the distance between the buffer size and batch size was further increased to 10,240 and 32, while the num layers hyperparameter reached the value 5. This training task consisted of at least 50 iterations in which many other hyperparameters ("learning rate", "epsilon", "lambd"), as well as rewards, were tested by trial and error, however, no further improvements of the end rewards were achieved.

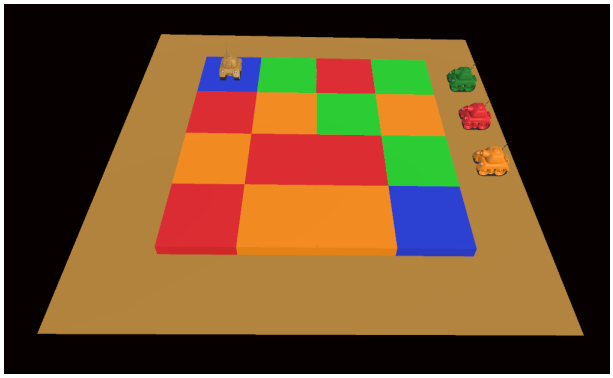

Figure 6: The four-by-four tiles environment

Figure 7 presents the rewards given to the AI agent in the four-by-four tiles environment. It can be observed that the rewards for the movement-related actions of the AI agent were further increased. The AI agent developed an unwanted behavior during training in which it would get stuck, without further performing any actions involving movement or dialogue. The solution provided was adding a new negative reward, which would be given to the AI agent if it reached a maximum of 10 steps without performing actions.

stepping on the red tile	-2
stepping on the green tile	2
stepping on the orange tile	0.1
stepping on the end tile	5
moving forward	-0.1
rotating	-0.1
dialogue action towards a spectator	0.1
going out of bounds/ into a barrier	-5
achieving the maximum rotations on a tile	-5
achieving the maximum number of no actions performed	-5
Maximum reward possible in an episode	13.2

Figure 7: Rewards for the four-by-four tiles environment

5 RESULTS

TensorBoard is a helpful tool that provides figures and histograms of the AI agent that was trained in Unity using the ML-Agents package. By using TensorBoard, we can visualize the evolution of the training of the AI agent and we can more easily understand the behavior that emerged after training. A great advantage of TensorBoard is that you can apply smoothing to the learning curves of the figures and in doing so, make them easier to interpret by reducing the amount of noise and fluctuations.

The figures below depict the best results of the AI agent observed when training in three drastically different environments: the two-by-two tiles environment, the three-by-three tiles environment, and the four-by-four tiles environment. A smoothing of 0.6 has been applied to all of the learning curves in the figures, but the learning curves used before the smoothing process have been kept and can be seen as a less visible line, as they are not the main focus of the evaluation and can be ignored. There are two different figures for each environment, one depicting the cumulative reward over time and the other showing the change in the length of the episodes over time.

The two-by-two tiles environment presents besides the AI agent, the green and the red agent. In Unity, there were added 8 more two-by-two tiles environments used in order to accelerate the training. Figure 8 highlights the mean reward that the nine AI agents managed to collect during the training process. The hyperparameter max_steps was set with a value of 500,000, allowing the AI agent to train for 38 minutes. The learning curve is almost straight, which indicates that the AI agent trained continuously and effectively. The maximum reward for this environment was 6.2, and the AI agent achieved a 5.8 reward at the end of the training. When looking

at the displayed behavior after training, the AI agent chooses the shortest and most rewarding route to the end tile. In terms of communication, the AI agent participates in dialogues with the spectators after every action, but it does not have a preferred agent to talk to.

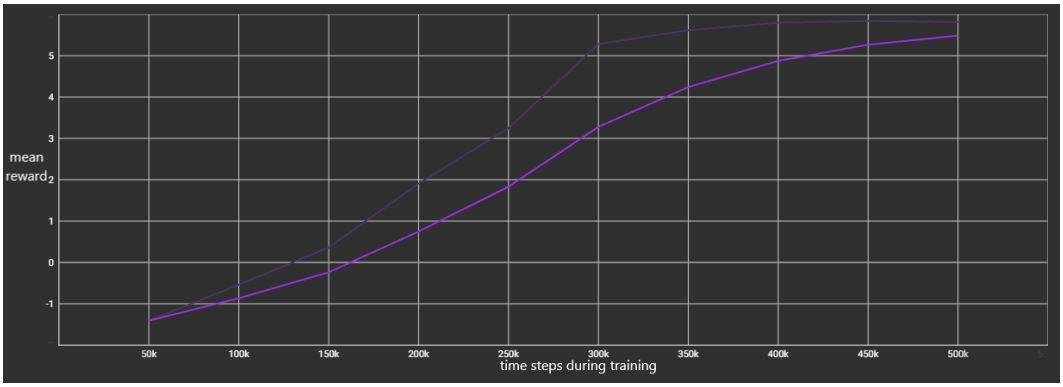

Figure 8: Two-by-two tiles environment - cumulative reward over time

Figure 9 presents the change in the episode length during training. At first, the AI agent starts to increase its number of steps after each iteration in order to explore the environment. After a certain threshold, between 28 and 29 steps, was crossed, the AI agent stopped exploring and started to optimize their actions and choose the best set of actions in order to complete the game with the biggest cumulative reward possible.

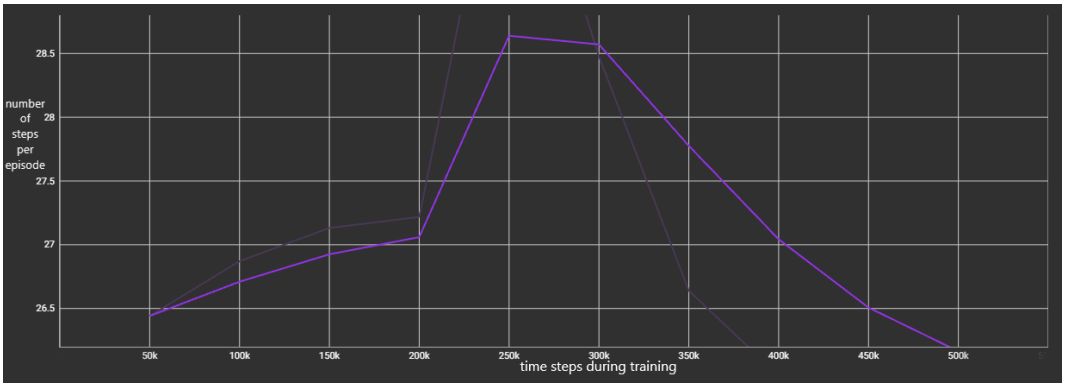

Figure 9: Two-by-two tiles environment - episode length over time

In the three-by-three tiles environment, the orange agent is added in order to observe a more complex communication between the agents. Eight other three-by-three tiles environments are present as well. In Figure 10, the mean rewards obtained by AI agents during training are revealed. Considering that this environment is more complex, the possible cumulative reward given in an episode can be much lower than in the previous

environment. It is shown that the max_steps hyperparameter has a value of 2,000,000, which gives the AI agent more steps to get better rewards and allows it to train for 94 minutes. The maximum cumulative reward achieved by the AI agent was 6.2, while the maximum possible cumulative reward in an episode was 8.4. We can observe a big difference between the achieved and potential rewards, as well as in terms of behavior. On one hand, in one episode, we can observe the AI agent choosing the shortest route, which is not the most rewarding route, but on the other hand, in the next episode, it chooses a completely unrewarding path that leads it outside the environment. In terms of communication, the AI agent chooses to speak mostly with the green agent and sometimes with the orange agent, resulting in a positive outcome as the AI agent differentiates the spectators.

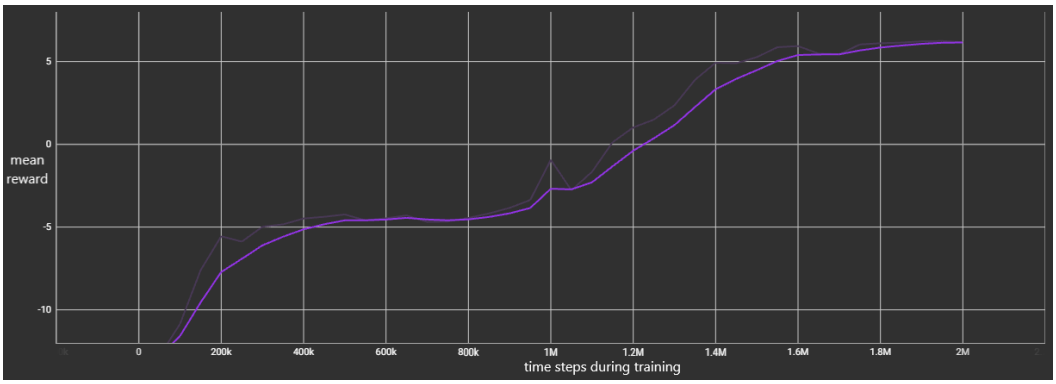

Figure 10: Three-by-three tiles environment - cumulative reward over time

As presented in Figure 11, the AI agent in the three-by-three tiles environments explored it at first and then exploited it by focusing on getting a bigger cumulative reward. The double spikes indicate the indecisiveness of the AI agent when presented with a more complex environment with a significantly larger number of possible states and actions.

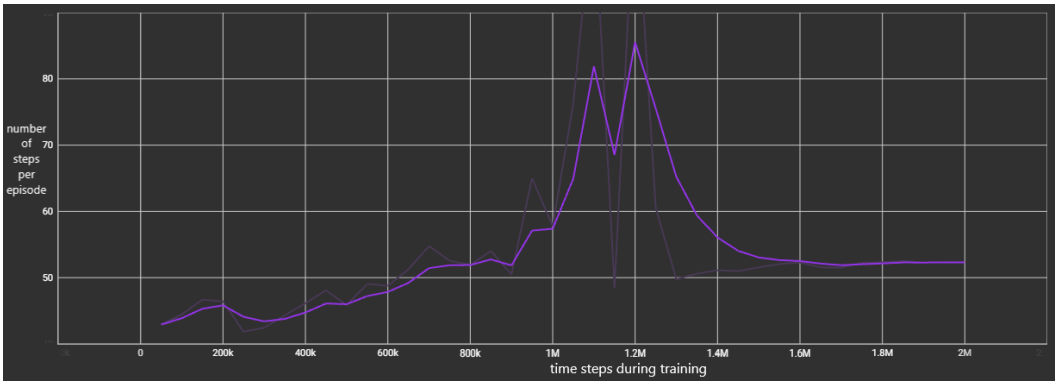

Figure 11: Three-by-three tiles environment - episode length over time

Figure 12 showcases the mean reward of the AI agents that trained in the four-by-four tiles environments. As it can be seen, the mean reward value started at the highest point of the learning curve and then only decreased, showing that the AI agents were unable to properly train or learn. Due to the fact that the mean reward of the AI agent only decreased, the learning process was stopped around the 1,600,000 step mark as it was deemed futile to continue the training, which lasted for only 103 minutes. The maximum cumulative reward achieved by the AI agent was -8.6, while the maximum possible cumulative reward in an episode was 13.2. Due to the big gap between the two cumulative rewards, the AI agent can be seen behaving randomly after training. The action choices of the AI agent are not clear as it can be observed moving in the environment without purpose. In terms of communication, the AI agent talks more to the green agent, but it looks like it does not really differentiate between the spectators as it also engages in dialogues with the orange and red agents.

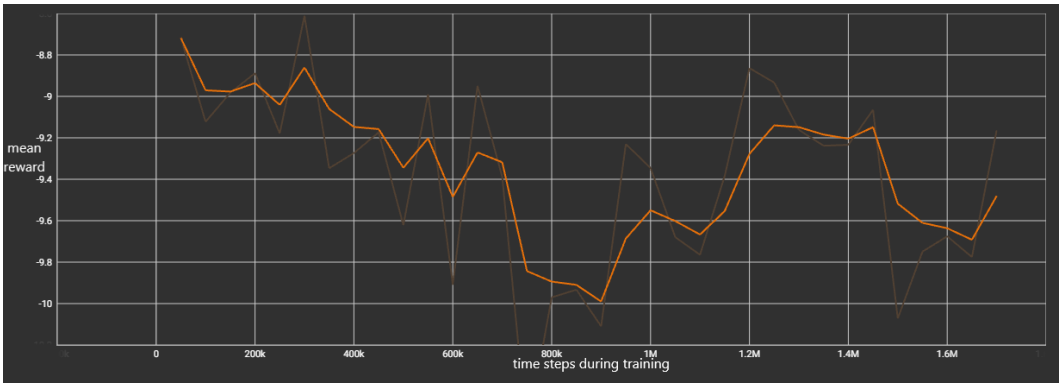

Figure 12: Four-by-four tiles environment - cumulative reward over time

In terms of the number of steps taken by the AI agent per episode in the four-by-four tiles environments, with the help of Figure 13, it seems like the AI agent is stuck in the exploring stage of the training, because the number of steps only increases. Considering that the AI agent is not arriving at the exploitation stage of the training, it is unable to complete the game or to find the shortest route in the environment or not even the path with the highest reward.

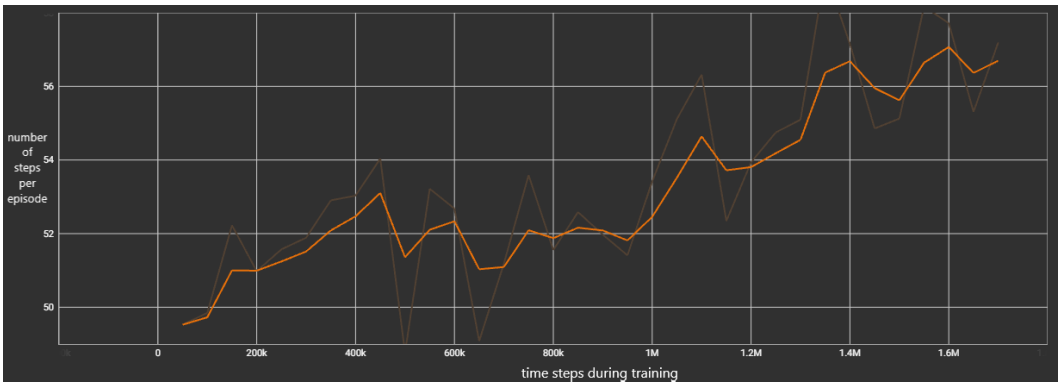

Figure 13: Four-by-four tiles environment - episode length over time

6 DISCUSSION

The goal of this project is to present the training of an AI agent in different environments and to research the communication between AI-oriented and non-intelligent agents. The first part of the goal centers around helping the AI agent to develop a behavior, that will enable it to achieve the maximum reward possible in a given environment. The second part of the goal is to observe the communication process between the agents, to determine if the different information given by the spectators will help the AI agent in differentiating them. This fact can be observed after training when looking at which non-intelligent agent does the AI agent chooses to speak to.

In terms of achieving the maximum reward, the AI agent situated in three different environments reached diverse conclusions. Firstly, in the two-by-two tiles environment, the AI agent managed to produce the desired behavior by choosing actions that resulted in the maximum cumulative reward possible. Secondly, in the three-by-three tiles environment, the AI agent chose after training the shortest path possible to the end tile and not the one that would give it the highest reward. Moreover, it could not maintain this behavior as it also wandered around the environment aimlessly, showing an exploratory behavior. Thirdly, the AI agent situated in the four-by-four tiles environment did not show any sign of wanting to arrive at the end tile and continued to apply its exploratory behavior after training. This confirms that given the right rewards and hyperparameters and with the help of spectator agents, the AI agent can learn to go through the environment to achieve a goal such as getting the maximum reward possible, a disadvantage to the learning process being the increased complexity of the environment.

When looking at the communication goal in this project, there are also three different outcomes coming from the agents interaction in the

environments. Firstly, in the two-by-two tiles environment, the AI agent interacts with both the green and orange agents. There is no preferred spectator with which the AI agent would wish to communicate and so either the environment is too simple for the AI agent to develop such a behavior or it needs even the poorer information from the red agent to go through the environment. In the three-by-three tiles environment, it is observed the tendency of the AI agent to mostly speak to the green agent, in so recognizing that the green agent gives the best information about the environment. Lastly, in the four-by-four tiles environment, the AI agent communicates more with the green agent but also talks frequently with the other spectators, leading to the belief that the AI agent is only slightly inclined to choose the green agent as the best informant. These three observations conclude that the AI agent can differentiate the spectators and can understand which one provides the best information, an interfering element being again the complexity of the environment and a deciding factor could be as well the number of spectators, however, further research is needed.

A visible limitation to this project was the knowledge available in articles about Unity and Ml-Agents, or more importantly, information about the hyperparameters and rewards that could be useful in training the AI agent. Apart from the articles made by Lukas et al. (2022) and Andersson (2021), there were not enough resources in the articles relating to, for example, how to properly use the hyperparameters. Most articles found and included in this project specified which hyperparameters were used in their own research, but did not give a reason as to why they were changed by being increased or decreased. Due to the training time being between 30 and 130 minutes and because over 20 hyperparameters are used, training proves to be a difficult task. A further direction of research would be to tackle this problem and provide a better understanding for future projects.

7 CONCLUSION

The research goals of this project were to help the AI agent in developing a behavior that will enable it to achieve the maximum reward possible in a given environment as well as to observe the communication process between AI agents and non-intelligent agents, to determine if the different information given by the spectators will help the AI agent in differentiating them. The results stated that the AI agent can learn to go through the environment to achieve a goal such as getting the maximum reward possible and that the AI agent can differentiate the spectators and can understand which one provides the best information. These results can

be placed in already existing research by further proving that there can exist communication between AI and non-AI agents as well as the fact that communication can be added to the learning process of an agent which has to discover or learn a new environment.

This project provides help in training AI agents in Unity using the ML-Agents package by providing context about the hyperparameters and rewards used as well as methods of visualizing the progress of the AI agents when they are learning. Unity and ML-Agents bring together the tools used in the gaming industry and in AI research. The communication-related knowledge presented in this research can be further used to improve games as well as coordination tasks, for example on a battlefield, in which an individual has to decide which source that he interacts with is the most reliable.

REFERENCES

Andersson, P. (2021). *Future-proofing Video Game Agents with Reinforced Learning and Unity ML-Agents.* Retrieved from https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1605238&dswid=-6111

Baby, N., & Goswami, B. (2019, 4). Implementing artificial intelligence agent within connect 4 using unity3d and machine learning concepts. *International journal of recent technology and engineering*. Retrieved from https://eprints.qut.edu.au/199647/

Botvinick, M., Ritter, S., Wang, J., Kurth-Nelson, Z., Blundell, C., & Hassabis, D. (2019, 5). Reinforcement Learning, Fast and Slow. *Trends in Cognitive Sciences*, *23*(5), 408–422. Retrieved from https://doi.org/10.1016/j.tics.2019.02.006 doi: 10.1016/j.tics.2019.02.006

Cao, Z., & Lin, C. (2019, 2). *Reinforcement Learning from Hierarchical Critics.* Retrieved from https://arxiv.org/abs/1902.03079

Cao, Z., Wong, K., Bai, Q., & Lin, C. (2020, 1). *Hierarchical and non-hierarchical multi-agent interactions based on unity reinforcement learning.* Retrieved from https://opus.lib.uts.edu.au/handle/10453/147184

Dorri, A., Kanhere, S. S., & Jurdak, R. (2018, 4). Multi-Agent Systems: A Survey. *IEEE Access*, *6*, 28573–28593. Retrieved from https://doi.org/10.1109/access.2018.2831228 doi: 10.1109/access.2018.2831228

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018, 9). *Unity: A General Platform for Intelligent Agents.* Retrieved from http://arxiv.org/abs/1809.02627

Kirtay, M., Oztop, E., Kuhlen, A. K., Asada, M., & Hafner, V. V. (2022, 9). Forming robot trust in heterogeneous agents during a multimodal interactive game.. Retrieved from https://doi.org/10.1109/icdl53763.2022.9962212 doi: 10.1109/icdl53763.2022.9962212

Lukas, M., Tomičić, I., & Bernik, A. (2022, 3). Anticheat System Based on Reinforcement Learning Agents in Unity. *Information*, *13*(4), 173. Retrieved from https://doi.org/10.3390/info13040173 doi: 10.3390/info13040173

Mehta, N. D. (2020, 1). State-of-the-Art Reinforcement Learning Algorithms. *International journal of engineering research and technology*. Retrieved from https://doi.org/10.17577/ijertv8is120332 doi: 10.17577/ijertv8is120332

Mmisupport. (2023, 2). The Importance of Digital Communication on the Battlefield. *KWESST*. Retrieved from https://www.kwesst.com/blog/the-importance-of-digital-communication-on-the-battlefield/

Mwiti, D. (2023, 4). 10 Real-Life Applications of Reinforcement Learning. *neptune.ai*. Retrieved from https://neptune.ai/blog/reinforcement-learning-applications

Patacchiola, M., & Cangelosi, A. (2016, 9). A developmental Bayesian model of trust in artificial cognitive systems.. Retrieved from https://doi.org/10.1109/devlrn.2016.7846801 doi: 10.1109/devlrn.2016.7846801

Sukhbaatar, S., Szlam, A., & Fergus, R. (2016, 5). Learning Multiagent Communication with Backpropagation. *arXiv (Cornell University)*. Retrieved from http://arxiv.org/abs/1605.07736 doi: 10.48550/arxiv.1605.07736

Unity-Technologies. (n.d.). *GitHub - Unity-Technologies/ml-agents: The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents using deep reinforcement learning and imitation learning.* Retrieved from https://github.com/Unity-Technologies/ml-agents

Urmanov, M. D., Alimanova, M., & Nurkey, A. (2019, 12). Training Unity Machine Learning Agents using reinforcement learning method.. Retrieved from https://doi.org/10.1109/icecco48375.2019.9043194 doi: 10.1109/icecco48375.2019.9043194

Vanvuchelen, N., Gijsbrechts, J., & Pardalos, P. M. (2020, 8). Use of Proximal Policy Optimization for the Joint Replenishment Problem. *Computers in Industry*, *119*, 103239. Retrieved from https://doi.org/10.1016/j.compind.2020.103239 doi: 10.1016/j.compind.2020.103239

Wang, Y., He, H., & Tan, X. (2020, 8). *Truly Proximal Policy Optimization.* Retrieved from http://proceedings.mlr.press/v115/wang20b.html

Youssef, A. M., Missiry, S. E., El-Gaafary, I. N., ElMosalami, J. S., Awad, K., & Yasser, K. (2019, 10). Building your kingdom Imitation Learning for a Custom Gameplay Using Unity ML-agents.. Retrieved from https://doi.org/10.1109/iemcon.2019.8936134 doi: 10.1109/iemcon.2019.8936134