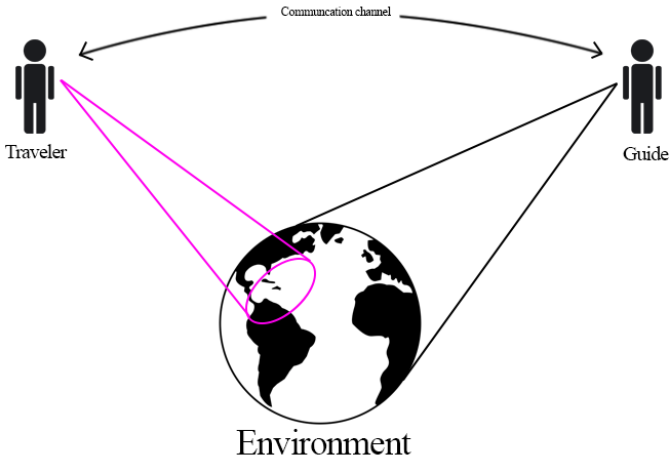

Efficient communication using reinforcement learning in a cooperative navigation game

Master's thesis in Computer science and engineering

Erik Bohman & Simon Rogmalm Hornestedt

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Efficient communication using reinforcement learning in a cooperative navigation game

Erik Bohman & Simon Rogmalm Hornestedt

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Efficient communication using reinforcement learning in a cooperative navigation game

Erik Bohman & Simon Rogmalm Hornestedt

Supervisor: Moa Johansson & Emil Carlsson, Department of Computer Science and Engineering
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A multi-agent system, including the two agents: traveler and guide, and their communication channel.

Typeset in LaTeX
Gothenburg, Sweden 2022

Efficient communication using reinforcement learning in a cooperative navigation game

Erik Bohman & Simon Rogmalm Hornestedt
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The thesis aims to investigate if agents are able to develop an efficient communication, with semantic meanings, and solve a navigation problem, using reinforcement learning. Additionally, it aims to evaluate the relevancy and benefit of one and two-way communication in comparison to each other and no communication.

The problem is tackled in a multi-agent system (two agents), using a cooperative navigation game. The agents possess different privately held information, they are hence equipped with a communication channel and a language with no initial semantic meaning to convey the information to each other and solve the task of finding a target inside an environment with distracting obstacles. The experiments take place in both a discrete and a continuous setting with a varying number of communication ways and are evaluated based on the average time to complete the navigation.

It is shown in the thesis that the agents can develop a language with a semantic meaning, which contributes to an efficient communication when set in a discrete environment and in a continuous static environment. However, it is inconclusive whether there are any significant benefits to be gained from a two-way communication compared to a one-way communication and whether the task can be solved in a continuous non-static environment.

Keywords: machine learning, reinforcement learning, efficient communication, multi-agent system

Acknowledgements

We would like to thank the Department of Computer Science and Engineering at Chalmers for the education provided over the years. A special thank you goes out to our supervisors: Moa Johansson and Emil Carlsson, for all the help and insights over the duration of the thesis, without your guidance we would have been stuck for a long time.

Finally, we would also like to thank our examiner Devdatt Dubhashi for the helpful ideas given at half-time, that ultimately lead to the grid world from which some conclusive results were extracted.

Erik Bohman & Simon Rogmalm Hornestedt, Gothenburg, June 2022

Contents

List of Figures

List of Tables

Acronyms

DRL Deep Reinforcement Learning. 5, 6

LogSoftmax Logarithmic Softmax. 15

MAS Multi-Agent System. 1–3, 6, 11, 12, 39, 40

MDP Markov decision process. 5, 6

PPO Proximal Policy Optimization. 5, 8, 9, 24, 33, 40

ReLU Rectified Linear Unit. 15

RL Reinforcement Learning. 1–3, 5, 6, 8

SGD Stochastic Gradient Descent. 9, 24

TRPO Trust Region Policy Optimization. 8

1

Introduction

When solving a real-world task, including communication, humans tend to use pragmatic and abstract concepts to facilitate and expedite arriving at the solution. The human language is one of the main reasons for our success throughout history. W. P. McCarthy et al. [15], describe how humans invent and learn to use abstract concepts for collaborating in a cooperative referential building game, where one agent gives instructions and the other acts as a builder. By forming concepts, humans increase information density while decreasing the length of each utterance.

Different cooperative reference games have been used in Multi-Agent System (MAS) machine learning [2, 6, 17], both games including and excluding communication. These games build on a setting where there are at least two or more agents, one acts as a sender of information and the other(s) as a listener.

This presents a setting: what if all agents in a MAS act as both sender and listener with different private information about the task, we framed this as a navigation game where two agents interact with each other. This poses a coordination task of communicating privately held information by the agents from one to the other, to efficiently solve a navigation game. Our study aims to investigate whether agents in a Reinforcement Learning (RL) system can develop a mutual language for a cooperative navigation referential game using two-way communication.

In this paper, we use a system with two agents: a guide and a traveler. The traveler interacts with the environment and has the ability to observe a limited field of view of its surroundings, meanwhile the guide possesses all information in the environment except the traveler's position. They are together given the task of making the traveler reach a target in the environment, and sometimes to also avoid distracting obstacles. The agents are equipped with a language that has no initial semantic meaning, and in order to be efficient, the agents need to agree on some semantics. An example of what the setting looks like can be seen in figure 1.1.

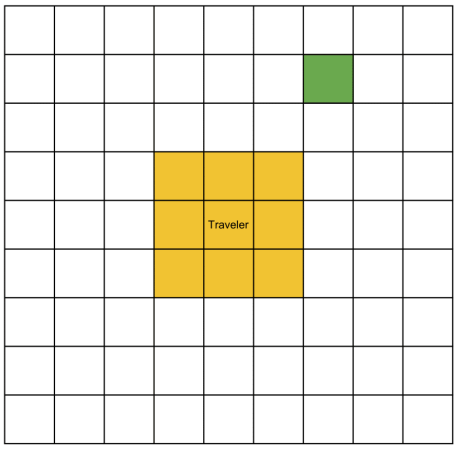

Figure 1.1: An example environment of the navigation game, the green tile is the target of the episode. The guide is able to observe the entirety of the white, yellow and green tiles. Meanwhile the traveler, located and the central tile of the environment, can only observes the yellow tiles, and move around the environment.

1.1 Goal

The goal of the thesis is to investigate if it is possible for agents in a multi-agent RL setting to develop a mutual language that will be used to communicate efficiently in a navigation task. The agents will be provided with a language that has a set of words. These words on the other hand have no semantic meaning. This is something the agents have to figure out on their own. In extension, a minor objective of this goal is to evaluate whether a two-way communication outperforms a one-way communication.

1.2 Research questions

- Given a language where words have no initial semantic meaning, is it possible for agents to develop a mutual language with semantic meaning in a MAS using RL in a cooperative navigation game?
- Does a two-way communication facilitate or expedite the solution of the task, or does a one-way communication achieve similar results? How does the one-way communication compare to no communication?

1.3 Hypothesis

We hypothesize that the agents will form a language that facilitates finding the solution to the task at hand. We also believe that, in contrast to human communication, the semantics of a word can differ when "speaking" or "listening", since the agents attempt to convey different information to each other, and do not require having a

mutual understanding of the word. They will form their definition of the language that is suited for their task.

Also, we hypothesize that, in a simpler environment (few distracting obstacles), the two-way communication is going to slow down the training but not affect the final result. Furthermore, in an increasingly complex environment (more distracting obstacles), two-way communication will be of increasing importance, since the task might not even be solvable without exchanging information held by both the agents. In environments where the task is solvable with both one-way and two-way communication, we hypothesize that the two-way communication speeds up the training.

1.4 Related work

In a paper from 2020, M.Kågebäck et al. show that RL can be used as an approach to successfully reach an agreement on a partitioning of a semantic space [11]. In their work, they have a two agents playing a color game, with the goal of partitioning the color space efficiently, in which the agents utilize speaker-listener communication over a noisy channel. Additionally, an approach taken by I.Kajic et al. [12], were successful in partitioning the space of a grid world, when attempting to solve a similar task as the one in this work, using a sender-listener communication. Although the work in this paper differentiate from these two mentioned, in its attempt to discover efficiency benefits from a second way of communication where both agents act as both sender and listener.

Adding a communication channel between agents in a MAS, where they solve tasks that requires coordination, enables the agents to develop communication [4, 12, 13]. Although, measuring and evaluating the communication that has emerged, is not a trivial task. R. Lowe et al. [14] survey the existing metrics proposes, they show that agents tend to send messages with a "meaning" (positive signaling), but refuse to listen to them (negative listening), which as a negative impact on the performance. However, they also propose a new metric, the causal influence of communication, where it's shown that agents that are trained with a causal influence as an additional reward performed better.

2

Theory

Throughout the work, Deep Reinforcement Learning (DRL) (see section 2.1.1), is used as a computational tool to allow the agents a trial and error style of learning. In section 2.1.2, we outline the algorithm that is used for policy optimization in the discrete world of section 3.4. An explanation to the theory of the Proximal Policy Optimization (PPO) trainer used in section 3.5, is given in section 2.1.3, this continuous world is built in the game engine Unity (see section 2.2).

2.1 Reinforcement learning

A Reinforcement Learning (RL) system has two main components, an environment and an agent [10]. The most typical form of the environment is a Markov decision process (MDP) [3], which is defined by the 4-tuple: (S, A, P_a, R_a). S is the state space (the different visible states in the environment). A is the action space (the available actions). $P_a(s_{t+1}, s_t) = Pr(s_{t+1}|s_t, a_t)$, the probability that action a in state s at time step t, will lead to state s_{t+1}. $R_a(s_t, s_{t+1})$ is the immediate reward of transitioning from state s_t (old state) to state s_{t+1} (new state).

The goal of the agent is to maximize its total reward (return/gain). The agent learns by collecting experience (rewards) in the environment. It observes its current state and makes an action, for then getting a reward and possibly ends up in a new state. The agent's experience could be gained by either exploration, making random actions to possibly gain total new experience, or by exploitation, taking an action based on its experience. Instead of using P_a, the agent uses the policy π, which decides how the agent exploits and/or explores its observed state and is a mapping from S to A. The described system is found in figure 2.1.

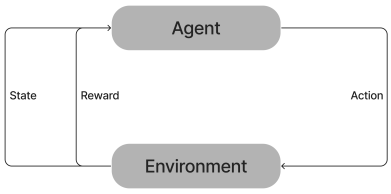

Figure 2.1: The architecture of a reinforcement learning system.

If there are multiple agents in a RL system, it is called a MAS. The agents in this system are not forced to have the same policy or the same problem to solve. However, in this thesis, the two agents have the same problem, a cooperative navigation problem, to solve, but they don't share the same policy. Additionally, the agents' perception of the environment in this thesis, takes the form of a Partially Observed MDP [22] since the traveler has a limited view, and the guide has no view of the traveler.

2.1.1 Deep reinforcement learning

Deep Reinforcement Learning (DRL) [8] is the area where deep learning is applied in reinforcement learning. Deep learning makes use of neural networks, multiple layers of neurons succeeding each other. The layers themselves can be thought of as a non-linear transformation in the form of weights, the weights are updated throughout training to allow the network to learn how to represent different abstractions. By using the network's transformations, the network maps an input to an output.

When using DRL the input of the network is the observations that the agents make at each time step, and the output at the last layer is a probability distribution of the action space, which in turn is sampled to get a single action. An example of this network architecture is found in the following figure 2.2, where one input layer is mapped using two consecutive fully connected layers to an output layer, this network architecture is the one used throughout this work.

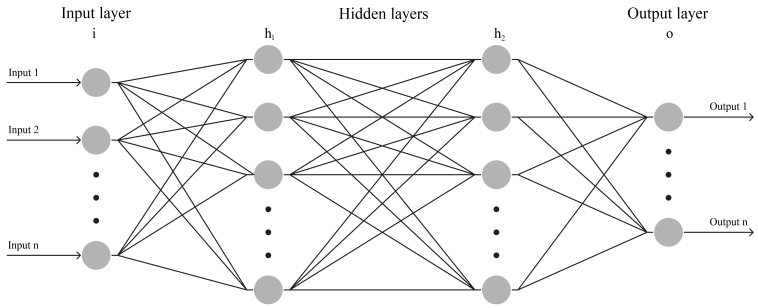

Figure 2.2: The architecture of a neural network with two hidden layers.

Powerful tools such as Tensorflow [1] and PyTorch [19] have been developed to enable programmers to easily create and train neural networks.

2.1.2 REINFORCE algorithm - with baseline augmentation

REINFORCE [21] is a policy-gradient algorithm, which makes use of gradient ascent when updating the policy π_θ, where θ is the weights of the neural network. The neural network, is the policy that the agent follows, the neural network takes a state as input and outputs a probability distribution of taking the different possible actions. At each time step of an episode, we collect the state, action and reward to

be able to define the trajectory τ, which is found in equation 2.1. The horizon of an episode is defined as H.

$$\tau = (s_0, a_0, r_1, s_1, a_1, ..., a_H, r_{H+1}, s_{H+1}) \tag{2.1}$$

We define G_t as the return at time step t, by the following equation:

$$G_t = \sum_{k=t+1}^{H+1} \gamma^{k-t-1} r_t \tag{2.2}$$

The goal of REINFORCE is to maximize the expected return over trajectories. The return R for the trajectory τ is defined as:

$$R(\tau) = (G_0, G_1, ..., G_t) \tag{2.3}$$

We define the expected return, $U(\theta)$, as:

$$U(\theta) = \sum_{\tau} \mathcal{P}(\tau; \theta)(R(\tau) - \beta(\tau)) \tag{2.4}$$

where we augment it with a baseline, denoted $\beta(\tau) = (\beta_0, \beta_1, ..., \beta_t)$, where β_t is the averaged return at time step t of the last 1000 episodes. $\mathcal{P}$ is the probability distribution of each possible trajectory. The probability relies on the neural network that is used. The update step used in backpropagation is defined as:

$$\theta \leftarrow \alpha \nabla_\theta U(\theta) \tag{2.5}$$

where α is the learning rate and ∇ is the gradient.

With these notations, the algorithm is formalized as 1.

Algorithm 1 REINFORCE

while not converge **do**
  Collect τ using policy π_θ
  Estimate R for τ
  Use τ to estimate $\nabla_\theta U(\theta)$
  Update weights θ for the policy π_θ
end while

The REINFORCE algorithm is used for policy optimization throughout the training of the grid world.

2.1.3 Proximal Policy Optimization

There is a multitude of approaches to policy optimization that have been proposed when it comes to RL with neural network function approximation, e.g. we have Deep Q-learning, trust region / neutral policy gradient methods and also "vanilla" policy gradient methods [20].

In Trust Region Policy Optimization (TRPO), one utilizes an objective function (referred to as "surrogate" objective) that is maximized under a constraint.

$$\max_{\theta} \hat{\mathbb{E}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}(a_t|s_t)}}\hat{A}_t\right] \tag{2.6}$$

$$\text{constraint: } \hat{\mathbb{E}}_t\Big[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\Big] \leq \delta \tag{2.7}$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function at time t, $\hat{\mathbb{E}}_t[...]$ and δ is a bound on the size of the policy update. While the TRPO approach is widely adapted, it does have some shortcomings when it comes to scalability and robustness due to being complicated.

To reduce the complexity of the implementation, J. Schulman et al. [20] introduce a different surrogate objective, namely a clipped penalty based approach:

$$\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} = r_t(\theta) \text{ such that } r_t(\theta) = 1 \text{ when } \pi_\theta(a_t|s_t) = \pi_{\theta_{old}}(a_t|s_t)$$

and define the objective as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{2.8}$$

by removing the constraint on the size of the policy update and instead penalizing the changes that move $r_t(\theta)$ away from 1 [20], where epsilon is a hyperparameter that is tuned to the task and environment.

In the same study in 2017, J. Schulman et al. [20] formalized an algorithm that alternates between sampling a policy and optimizing based on the samples. The algorithm is called Proximal Policy Optimization (PPO) and is a family of policy optimization methods that benefit from the stability and reliability of trust region methods but are simpler to implement. When used in neural networks, one makes use of the learned state-value function $V(s)$ and computes a loss function constructed from a combination of the surrogate policy and value function error term. The objective for this approach is ultimately constructed as:

$$L_t^{VF} = (V_\theta(s_t) - V_t^{target})^2 \tag{2.9}$$

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\Big[L_t^{CLIP}(\theta) - c_1 L_t^{VF} + c_2 S[\pi_\theta](s_t)\Big] \tag{2.10}$$

Where L_t^{VF} is the squared value function loss, S an entropy bonus used to augment the objective further and c_1, c_2 are coefficients.

Algorithm 2 uses a fixed-length trajectory segment where in each iteration N parallel actors collect T samples of data. This creates the requirement of an advantage estimator that does not consider time-steps beyond T. In [20] a generalized advantage estimation is used that satisfies the requirement and generalizes a previous estimator that was suggested by V. Mnih [16] in 2016.

$$\begin{aligned} &\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \\ &\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{2.11}$$

The surrogate loss is then constructed and optimized using a gradient descent optimizer like Stochastic Gradient Descent (SGD) or Adam for K epochs.

Algorithm 2 Proximal Policy Optimization (PPO)

1: **for** iteration = $1, 2, ...$ **do**
2: **for** actor = 1,2,...,N **do**
3: Run policy $\pi_{\theta_{old}}$ in environment for T time steps
4: compute advantage estimates $\hat{A}_1, ..., \hat{A}_t$
5: **end for**
6: Optimize surrogate L w.r.t. θ with K epochs and minibatch size $M \leq NT$
7: $\theta_{old} \leftarrow \theta$
8: **end for**

The PPO algorithm implemented in the toolkit (see section 2.2.1) using the clipped version, and is utilized in this works continuous world.

2.2 Unity

Unity[1] is a game engine used for real-time game development for 2D, 3D, VR, and AR games. This game engine was developed by Unity Technologies and was released in 2005. The game engine includes both a rendering and physics engine. For the engine to be user-friendly, Unity Technologies has developed a user interface that is called the Unity Editor. Unity was initially released for MAC OS X users only. Nowadays, Unity editor is supported on Windows, macOS and Linux. The engine, on the other hand, has more than 19 different supported platforms.

Unity uses an implementation of Mono[2] for scripting. Every Unity script derives from the base class MonoBehaviour, this class includes some event functions that are executed depending on the trigger event, these can be found in the documentation[3].

[1] `https://unity.com/`
[2] `https://www.mono-project.com/docs/advanced/embedding/scripting/`
[3] `https://docs.unity3d.com/ScriptReference/MonoBehaviour.html`

2.2.1 Unity Machine Learning Agents toolkit

The Unity Machine Learning Agents Toolkit [4] (ML-Agents toolkit) [9] is a package for Unity. This package is an open-source project that enables games and simulations to serve as environments for training intelligent agents. The implementations, which are based on PyTorch [18], offer state-of-the-art algorithms to enable us as developers to easily train intelligent agents.

The behavior of the agents, is implemented by making use of the class Agent, which is an extension of the MonoBehaviour class. An important function in this class is the CollectObservations function, this function collects observations that will be the input to the neural network that is trained. However, it is the programmer's task to decide what observations that should be added. This function, how the agent is rewarded, and many more can be found in the documentation[5].

2.3 Linguistics - efficient communication

We use the term efficient communication in the sense that it is a communication which uses a language (with no initial semantic meaning) that has emerged to contain semantic meaning for each agent. For the communication to be efficient, it also has to significantly improve or expediate the agents' solution in comparison to no communication.

According to K. Cao et al. [4], a necessary prerequisite for communication to emerge, is that a task is needed that requires coordination between multiple agents. Where none of the agents possess all the information to solve the task individually, and also a communication channel where the agents can communicate with messages.

[4] `https://github.com/Unity-Technologies/ml-agents`

[5] `https://docs.unity3d.com/Packages/com.unity.ml-agents@2.3/api/Unity.MLAgents.Agent.html`

3

Method

The problem is tackled in an MAS in the setting of a cooperative navigation game. This particular system consists of two agents, one named the *guide* (see 3.3.1) and the other the *traveler* (see 3.3.2). The traveler's task is to navigate in an environment and reach the *target* (see 3.1), although there are obstacles out in the environment to make this harder, known as *distractors* (see 3.1). The traveler has a limited vision, which is where the guide comes to the rescue; the guide resides outside the environment, but possesses a map of the environment. The guide's task is to help the traveler reach the target. To be able to do this, the agents are equipped with a *language* (see 3.2) and a *communication channel* (see 3.2.1). With this setup, they attempt to convey the information, held privately by the agents, to each other in order to solve the navigation task and find the target as efficiently as possible.

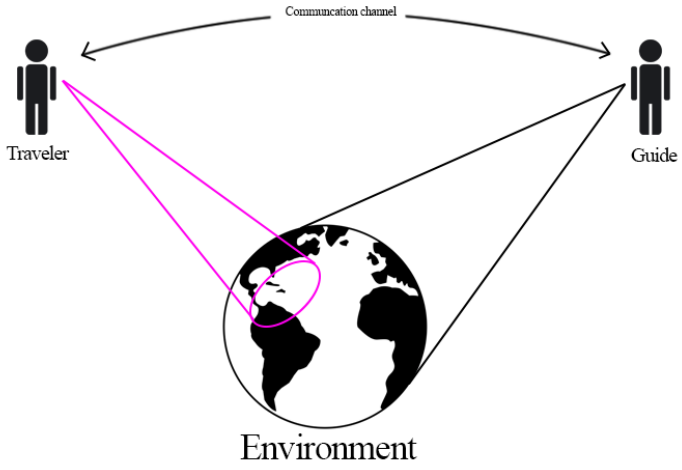

Figure 3.1: An abstraction of the described multi-agent system.

The experiments done in this thesis are performed in two different settings. The first setting is a grid world developed in Python [1], and the second setting is a continuous world developed in Unity (see 2.2) and the toolkit ML-Agents (see 2.2.1).

[1] `https://www.python.org/`

3.1 Environment

The environment is a flat surface, like a floor, where different objects can spawn at the start of each episode. The objects in the environment are referred to as *obstacles*. Obstacles have one property, it is either a target, which yields a positive reward when moving into it and terminates the episode, or a distractor, on which the traveler gets a possible negative reward and bumps back to its previous position when colliding with it, although the episode is not terminated upon contact with a distractor.

3.2 Language

The language $\mathcal{L}$ consists of distinct words w. The words in the language are mapped to one-hot encoded $|\mathcal{L}|$ or $|\mathcal{L}|-1$ sized vectors, depending on if we include the zero-vector $w_0 = [0, ..., 0]$ or not. Initially, the words have no semantic meaning before the training is commenced, meaning that the agents have to collectively decide on the semantic of each word. Additionally, the size of language $|\mathcal{L}|$ can be varied in different training settings and environment.

3.2.1 Communication channel

To allow the agents to communicate with each other, they utilize a communication channel. The agents can interact with the communication channel in two ways. They could either add a message to the communication channel (with a receiver), or they can collect messages that they are supposed to receive. The messages that the guide is supposed to receive in the communication channel are denoted m_g, and the traveler's messages are denoted m_t. The motivation for using a communication channel and not sending the message directly to the receiver is that the receiver could now read the message when it wanted to.

3.3 Agents

In the MAS, there are two different agents. The first one is the guide, which resides outside the environment, and the second one is the traveler, which resides inside the environment. The following two sections are going to describe these two agents more in detail.

3.3.1 Guide

The guide is an agent that possesses a complete map of the environment, but not the position of the traveler. With this setting, the guide's task translates to learning how to interpret the location of the target based on the map, and further to convey this information to the traveler, using the language and the communication channel given.

At each time-step, the guide polls the channel, and reads all the messages that are present in the channel at that time-step. Using two-way communication adds an extra layer to the observations for the guide. Since the traveler conveys information to the guide, this becomes part of the state observations. At each time-step, the guide polls the communication channel, and reads all the messages that are present in the channel at that time. With this additional information, the guide can attempt to interpret the location of the traveler, and therefore convey more useful information to the traveler.

3.3.2 Traveler

The traveler interacts with the environment, and can only observe what is closest to it. With each episode, it has no prior knowledge of the target nor the layout of the environment it is currently in.

With each time step, it takes an action that decides its next movement. This movement can be one of 5 different values, either go up, down, left, right, or do nothing. The traveler's task is to locate and interact with the target, to be able to terminate the episode and get a positive reward. Since the traveler can only observe its surroundings, the communication from the guide will be crucial for the traveler, for it to be able to grasp the entire environment efficiently during an episode. In extension, this also requires the traveler to interpret the sentiment of the guide.

Alongside interacting in the environment, and depending on the experiments, the traveler can communicate with the guide, using the same language as the guide. This will extend the guide's knowledge, and therefore the traveler might get better instructions from the guide in return.

3.4 Grid World

Several experiments were made in a discrete environment, a grid world with the size 9×9, an example of this environment can be found in figure 3.2. The width and height of the grid world remained the same throughout the experiments. However, what was inside the grid world varied, it started from a very simple setting, and gradually moved towards an increasingly complex world. The distractors in this grid world give no negative reward. Although, the traveler gets a small negative reward at each time step, and therefore the traveler wants to reach the target as fast as possible, and avoid the obstacles.

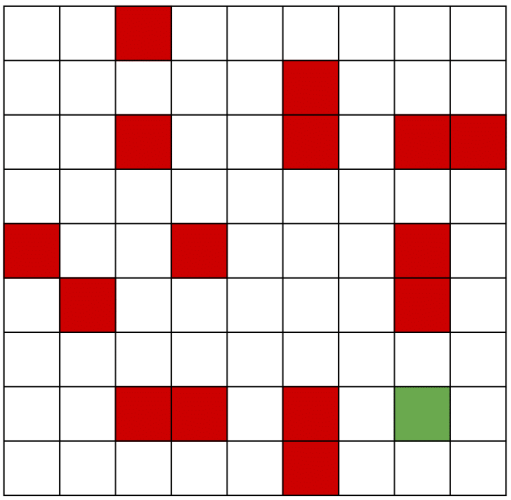

Figure 3.2: A grid world with the size 9×9. The green tile is the target and the red tiles are distractors.

The experiments in the grid world used the language $\mathcal{L} = w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7$, where w_0 is the zero-vector. Therefore, each vector has a size of $|L| - 1 = 7$.

The proposed idea of this project is to use a two-way communication, this might be hard to implement directly, and an approach where we gradually moved towards a two-way communication were used.

In the following sections, we start with explaining the observations for the guide and traveler, then dig deeper into the neural network that is used for these agents, and finally, three different experiments are described.

3.4.1 Observations

Both the guide and traveler utilize observations that are fed through their neural networks (see 3.4.2) to decide upon the next action. The guide is using one neural network for communicating, while the traveler is using two neural networks, one for communicating and one for moving in the environment. Therefore, the traveler utilizes two slightly different observations in each time-step when deciding between the two actions that it takes: communication and movement.

The guide's observation, O_g, consists of every tile in the environment, and a message, m_g, from the traveler (if the setting of the experiment allows this). The guide utilizes a neural network, NN_g, to choose an action, A_g, which is mapped to a word w in the language $\mathcal{L}$.

Moving to the traveler's first observation, O_{t_c}, used by the neural network, NN_{t_c}, to choose an action, A_{t_c}, which is mapped to a word w in the language $\mathcal{L}$. The traveler observes a 360° field of view that reaches one grid away from itself, including the grid that it is standing on at the moment. It can also observe its own x- and y-coordinates in the environment.

The second observation, O_{t_a}, is the one that is used for interacting with the environment. Here, the traveler has the exact same observations as for NN_{t_c}, additionally it also observers one message, m_t, from the guide (if the setting of the experiment allows this). The traveler uses the neural network, NN_{t_a}, to choose an action, A_{t_a}, which corresponds to a movement in the environment.

3.4.2 Neural Network

The observations explained in the previous section are fed into the input layer i. The neural network consists of two hidden layers, h_1 and h_2, both with the same size (denoted *hidden size*). Between the hidden layers, the activation function Rectified Linear Unit (ReLU) is used. The last layer is the output layer o, where the activation function Logarithmic Softmax (LogSoftmax) is used. The following figure 3.3 shows the described architecture. This architecture is used by all neural networks, although the size of the layers might vary between the experiments.

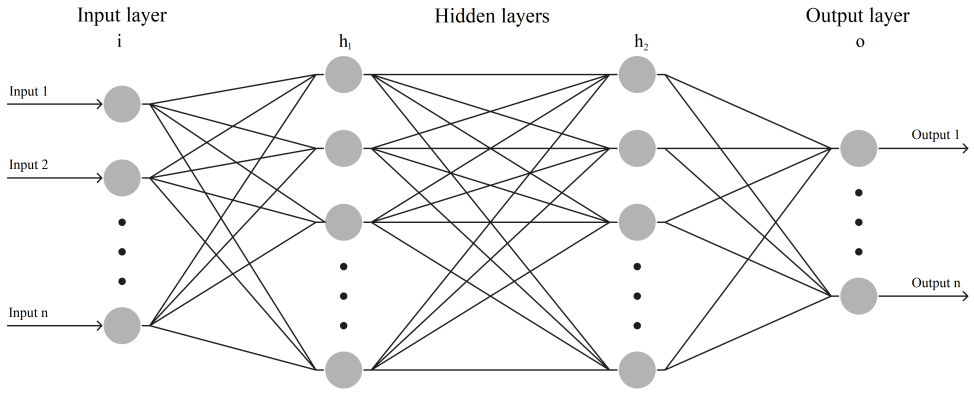

Figure 3.3: The architecture of the neural network that is used for both the traveler and the guide.

3.4.3 No communication experiment

First, we conducted an experiment to verify that our setup worked, and to verify that the traveler was able to navigate and search smartly. Therefore, no communication was used in this approach.

An environment as simple as possible was chosen here. In this environment, no distractors spawn. Although, one target spawns on any of the tiles in the environment, except on the traveler's starting position. An example of the environment, and a depiction of when the traveler can see the target and not, is found in figures 3.4 and 3.5. The idea behind using a simple environment is that there is no need to use a complex one when attempting to verify that the traveler can navigate smartly, a more complex environment would simply just slow down the training.

In algorithm 3, we find the training loop used in the experiment. At the start of an episode, a new environment is generated with a randomized target position. The

traveler is initialized to start at the midpoint of the environment. Lines 5-6 shows a step of an episode. With each step the traveler collects O_{t_a} and feeds it through NN_{t_a}, to choose A_{t_a}, this action is mapped to a movement in the environment using the deterministic function $move(A_{t_a})$, if the goal is reached or the maximum amount of steps for an episode has passed, after the step, the episode is terminated. In this experiment, the REINFORCE algorithm from section 2.1.2 is used and the collected steps of each episode function as the trajectory τ in algorithm 1.

This experiment was run twice with two different network settings to find a parameter tuning that could smartly and close to optimally solve the task. These settings differed only in the hidden sizes of the networks: the first test was run with a hidden size of 10 neurons, and the second with a hidden size of 16 neurons.

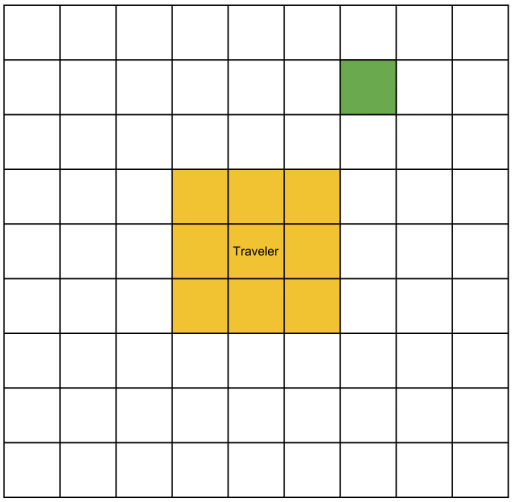

Figure 3.4: The traveler in an environment with no distractors and one randomly spawned target, that is colored green. The traveler's field of view is colored in yellow. The traveler can not see the target.

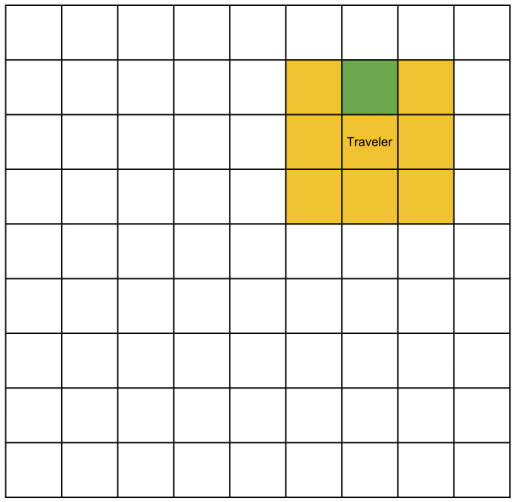

Figure 3.5: The traveler in an environment with no distractors and one randomly spawned target, that is colored green. The traveler's field of view is colored in yellow. The traveler can see the target.

Algorithm 3 Training Loop – No communication

1: **for** episode e = 1,2,...,N **do**
2: Generate new environment for the episode
3: Reset traveler to starting position
4: **for** step t = 1,2,...,T **do**
5: Collect O_{t_a} feed to NN_{t_a}, choose A_{t_a}
6: Step in environment $move(A_{t_a})$ and end episode if goal is reached
7: **end for**
8: Update weights of NN_{t_a} using REINFORCE, with episode τ
9: **end for**

This setting allowed us to estimate the number of steps needed to reach the target

(expected episode length), by simply traversing an optimal path (found in figure 3.6) without the given communication, with this we can formulate a benchmark for when the communication is beneficial to the solution of the problem.

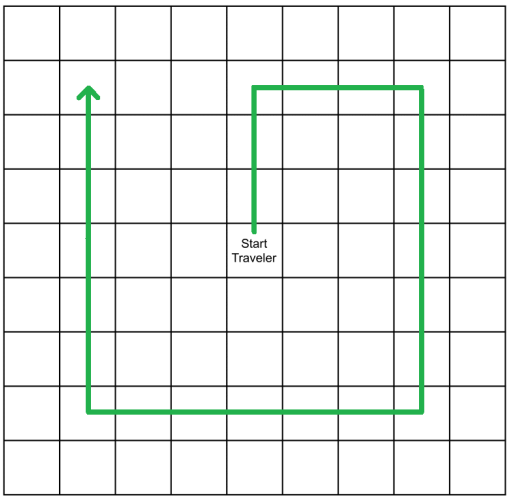

Figure 3.6: A theoretical optimal search path for an empty 9×9 grid world, where the target can spawn at any position, except the traveler's starting position.

We can formulate a random variable $\mathbf{X}$, the probability that the target is spawned in one of the visited squares at time t given that we walk an optimal path, and a random variable $\mathbf{Y}$, the number of optimal steps to the target after finding it.

To concretize $\mathbf{X}$. we know that there are 8 ways to move optimal in the environment (each optimal search path, isomorphic to the next with only directional change), with each step 3 more squares are visited, except for the first where the 8 surrounding squares of the traveler starting position are visited. This means the number of visited squares is equal to, $8 + 3x$ if x is the number of steps taken on an optimal path. Since the target can spawn in 1 out of 80 squares each episode, all squares are visited in 24 steps, and hence, we estimate the expected length of this to be.

$$\mathbb{E}[\mathbf{X}] = \sum_{x=0}^{24} \frac{1}{80}(8 + 3x) = 13.75 \tag{3.1}$$

For $\mathbf{Y}$ we know that the traveler can only move in cardinal directions (vertical or horizontal) and the target will always be discovered in either a diagonal or a cardinal square. Therefore, we estimate the expected length of $\mathbf{Y}$ by the following.

$$\mathbb{E}[\mathbf{Y}] = \frac{2}{3} \cdot 2 + \frac{1}{3} \cdot 1 = 1.67 \tag{3.2}$$

Since the two events are independent, we can simply just add the two values together, therefore the final expected episode length is

$$\mathbb{E}[\mathbf{X}] + \mathbb{E}[\mathbf{Y}] = 15.42 \tag{3.3}$$

3.4.4 One-way communication experiment

In this experiment, only one alteration was made: one-way communication was introduced. The guide was made able to send one word as a message at every time step, to the traveler via the communication channel. This experiment was done to verify if introducing a one-way communication would improve the traveler's efficiency of finding the target. In this setting, the guide observes the environment as described in section 3.4.1, although no communication from the traveler. This observation is found in figure 3.7.

Theoretically, the guide could now divide the area of the environment into $|\mathcal{L}|$ sectors, to be able to tell the traveler where to go, and therefore the traveler does not have to search through the whole environment to find the target. This has been shown in [12], although in their approach they send a one-hot encoded vector to the traveler at the start of the episode, and therefore do not have any continuous communication.

The training loop of the one-way communication is specified in algorithm 4. The additional communication that is added in this experiment is seen in lines 5-6. with each step of the episode, the guide collects O_g and selects A_g using NN_g, A_g is mapped to a word w of the language $\mathcal{L}$, which is passed as a message m_t to the traveler. In lines 7 through 12 the system proceeds as the system of the no communication experiments, although in line 10, we also see that the neural network NN_g is updated alongside NN_{t_a}.

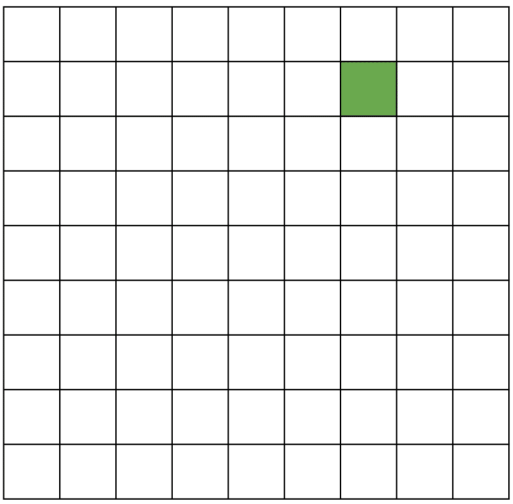

Figure 3.7: The guide's observation of an environment with no distractors and one randomly spawned target, colored green.

Algorithm 4 Training Loop – One-way communication

for episode e = 1,2,...,N **do**
    Generate new environment for the episode
    Reset traveler to starting position
    for step t = 1,2,...,T **do**
        Collect O_g feed to NN_g, choose A_g
        Send $\mathcal{L}(A_g) = w$ to traveler
        Collect O_{t_a} feed to NN_{t_a}, choose A_{t_a}
        Step in environment $move(A_{t_a})$ and end episode if goal is reached
    end for
    Update weights of NN_g, NN_{t_a} using REINFORCE, with episode τ
end for

This setting also allowed us to evaluate the performance by a calculated expected episode length. We formulate the random variable $\mathbf{X}$, such that it is the Manhattan distance to the target from the traveler's starting position, this is further referred to as, the distance. $\mathbf{X}$ takes the value i when the target is spawned at a tile that is located at i. Although the target's spawning position is uniformly distributed as $p = \frac{1}{80}$, there are a varying amount of squares at each distance. We denote the event that the target spawns at the distance of i steps by:

$$Pr(E_i) := p_i = pk_i \tag{3.4}$$

where k_i is the number of tiles at distance k_i, calculations of each of these expectations can be seen in 3.1.

Table 3.1: Values and probabilities for the random variable $\mathbb{X}$, used for calculating the expected Manhattan distance to the target, in the one-way communication experiments.

i	#Tiles at distance i, k_i	$Pr(E_{\mathbf{X}_i}) = p_i$
1	4	$\frac{4}{80}$
2	8	$\frac{8}{80}$
3	12	$\frac{12}{80}$
4	16	$\frac{16}{80}$
5	16	$\frac{16}{80}$
6	12	$\frac{12}{80}$
7	8	$\frac{8}{80}$
8	4	$\frac{4}{80}$

Since we now have the expected value that the target spawns at a distance i for all

possible distances i. By linearity of expectation, we know that

$$\mathbb{E}[\mathbf{X}] = \sum_{i=1}^{8} p_i i = 4.5 \tag{3.5}$$

The following figure 3.8 gives a visual representation of the Manhattan distance from each tile to the traveler's starting position.

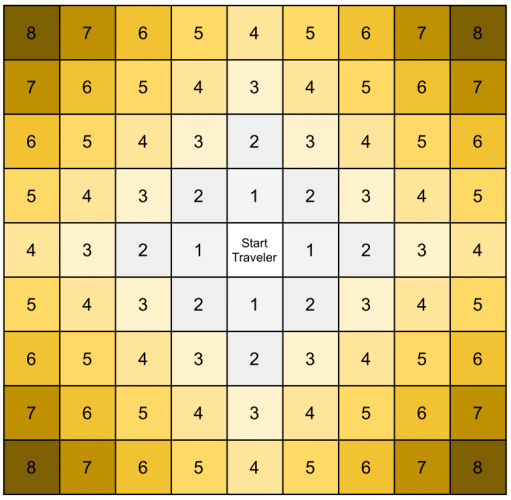

Figure 3.8: The Manhattan distance from each tile to the traveler's starting position.

3.4.5 Two-way communication experiment

In the third, and final experiment in the grid world, it was time to introduce the two-way communication. By introducing this, we needed a more complex environment, otherwise, the traveler would not have anything more to talk about, than when it sees the target; when the traveler sees the target, it would not make much sense to talk about it since the traveler could just go directly by itself to the target.

To make the environment more complex, we introduced distractors, in the form of walls, in the third and seventh rows of the grid world. At the start of every episode, three adjacent squares were selected in each wall of distractors to create a hole that the traveler could pass through. The target's position was randomized at the start of every episode, although it could not spawn at the position of a distractor or in the adjacent tiles of the traveler's starting position. An example of this grid world can be found in figure 3.9.

The training loop of the experiment is specified in algorithm 5. The difference from the previous experiments is seen in lines 5-6. With each step of the episode, the traveler starts by collecting O_{t_c} and selecting A_{t_c} using NN_{t_c}. A_{t_c} is then mapped to a word w of the language $\mathcal{L}$, which is passed as a message m_g to the guide. From line 7-12, the system behaves as a one-way communication experiment, with the

addition that the second network of the traveler NN_{t_c} is also updated using the same trainer as previously.

Algorithm 5 Training Loop — Two-way communication

for episode e = 1,2,...,N **do**
    Generate new environment for the episode
    Reset traveler to starting position
    for step t = 1,2,...,T **do**
        Collect O_{t_c} feed to NN_{t_c}, choose A_{t_c}
        Send $\mathcal{L}(A_{t_c}) = m_g$ to guide
        Collect O_g feed through NN_g, choose A_g
        Send $\mathcal{L}(A_g) = m_t$ to traveler
        Feed O_{t_a} to NN_{t_a}, choose A_{t_a}
        Step in environment $move(A_{t_a})$ and end episode if goal is reached
    end for
    Update weights of NN_{t_c}, NN_g, NN_{t_a} using REINFORCE, with episode τ
end for

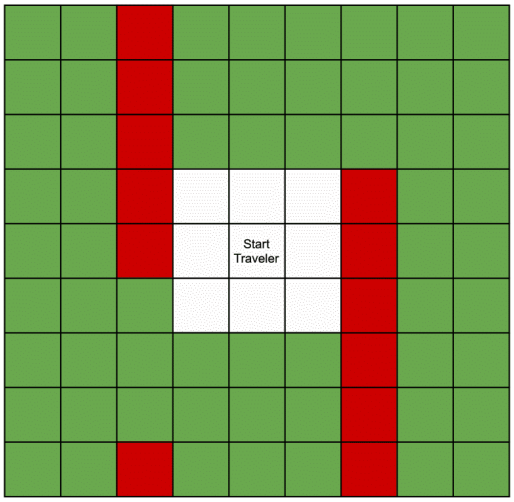

Figure 3.9: A generated environment of the grid world in the two-way communication experiment. The target can spawn on any of the green tiles. The red tiles are distractors.

Since an optimal path in such an environment was too hard to find, by theoretical means, we needed some kind of comparable baseline. To find this comparable, we chose to compare this approach to a one-way communication system trained in this more complex environment. By comparing these two approaches, we can see how beneficial the two-way communication is to the efficiency of the system. Both in terms of training time and average final episode length.

3.5 Continuous World

In the continuous world experiments, we chose to use an environment with four unique (and to the agents distinguishable) obstacles placed in perpendicular positions to the traveler's starting position at the midpoint.

The environment has two versions, it is either static or non-static. In the static version, the obstacle at each position have the same shape and color in every episode, although which one of the obstacles that are selected as the episode's target is randomized. An instance of a static environment, is found in figure 3.10. In the non-static version, the obstacle at each position can alter color and shape between the episodes, although all shape and color combinations are present in every episode and the target selection is made randomly. Figure 3.11 depicts the difference between two episodes in the non-static world.

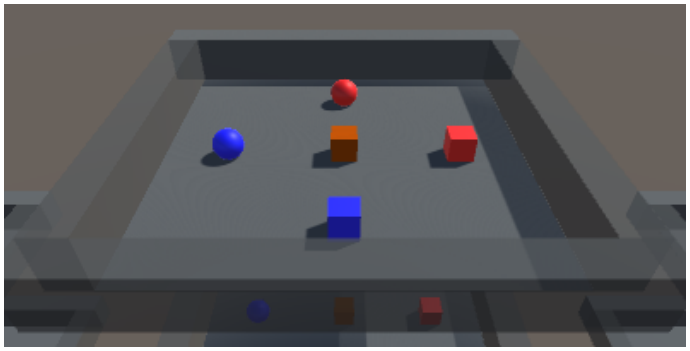

Figure 3.10: An example of the described static environment. The Orange cube is the traveler at its starting position for each episode. The four obstacles are the red and blue spheres, as well as the red and blue cubes.

Figure 3.11: An example of the difference between episodes in the non-static environment. The Orange cube is the traveler at its starting position for each episode. The four color and shape combinations spawn at different positions in each episode

The obstacles have the same property as previously, either it is a target or a distractor. In this continuous world, both the agents are rewarded unanimously. The reward is given if the traveler collides with an obstacle: if the obstacle is the target, the agents are given a positive reward and the episode is ended, if the obstacle is a distractor, the agents are given a punishment (negative reward) and the episode continues. The size of the reward from finding the target was scaled linearly decreasing with episode length, to give the incentive of finding a quick solution.

The language that the agents are using to communicate their observations, in the continuous world, consists of only five words $\mathcal{L} = w_0, w_1, w_2, w_3, w_4$. As mentioned in section 3.2 the vectors are one-hot encoded, with the size $|\mathcal{L}| - 1 = 4$, with w_0 being a zero-vector. The communication channel is used to pass the information to the peer agent within every fifth time step, due to the simulation speed of Unity. Each agent can send a maximum of one message per time step, hence each agent can receive a maximum of 5 messages every fifth time step. When the agents are reading their inbox, the agents will always read the latest message first. Each message m_j consists of one word w from the language $\mathcal{L}$, where j denotes the time-step that the message was sent. Since this environment is more asynchronous than the grid world, some safety precautions were implemented. The agents maintain a memory of the last 5 messages read, to not lose information passed between the agents.

To evaluate this world, we consider the episode length. Since there are only two ways to reset/end an episode: reaching the target or reaching the maximum number of steps allowed in an episode, one can consider the following relation: the shorter the episode length, the better the solution to the task. This world is also visually simulated, which can help understand the behavior of the traveler to assess whether it seems to behave somewhat deterministic or completely guessing.

3.5.1 Observation

The traveler observes its surroundings in the environment via a snapshot around itself at every time step, its field of view covers a space equal to a 3×3 grid. Figure 3.13 shows the disparity between the view for the traveler, which is colored in yellow, and the entire environment. In total, the traveler observes the two last snapshots, the messages in the communication channel, and a local two-dimensional vector of its position, this observation is denoted O_t.

The guide observes a snapshot, of size 9×9, that covers the entire environment, an example of this snapshot can be seen in figure 3.12. It also observes the integer value representing the target of the given episode. The entire observation is denoted with O_g

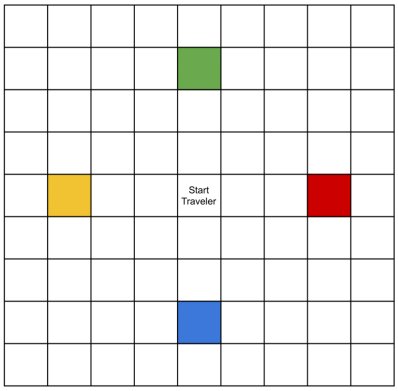

Figure 3.12: The observation for the guide, including four unique obstacles that is colored in red, green, blue and yellow.

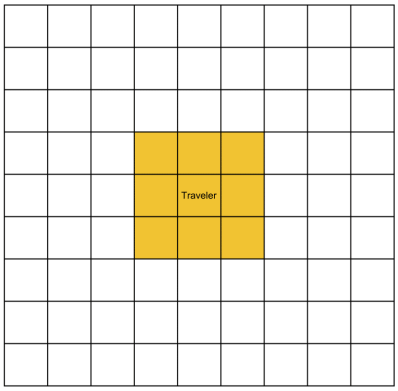

Figure 3.13: The traveler's field of view, a 3×3 square colored in yellow, in comparison to the entire environment.

3.5.2 Trainer- and Network settings

The Unity toolkit uses the PPO algorithm described in algorithm 2 to train the neural networks, the two neural networks are denoted NN_t and NN_g. The hyperparameters and network settings are chosen separately for each experiment. These networks are quite similar to the ones used in the grid world. They consist of two linear layers alike the previous ones, although the size of these layers are significantly larger than the one in the grid world, with a hidden size of 48 neurons. An argument was made that these networks needed increased size due to the continuous movement in the environment.

For the tuning of the PPO algorithm, the hyperparameter $\theta = (\epsilon = 0.1, c_2 = 0.01, \alpha = 0.001)$ is used in all the settings of the continuous. When running the policies in algorithm 2 (line 3), the agents collect their respective observations O_g, O_t and choose their actions using their neural networks, the PPO trainer collects the rewards over the time horizon $T = 512$ and computes the advantage $\hat{A}_t$ for all t, the optimization is then run with a batch size of 64 over 10 epochs, using SGD. Since the maximum episode length of the system is set to 1500 time steps, at least one full episode is collected in a time horizon.[2]

3.5.3 Continuous world experiments

The experiments were commenced using one-way communication. To make sure that the information in the environment was sufficient to solve the task at hand, an approach was taken where a series of tests were performed. First, the system was trained in a static environment (see section 3.5.3.1) to establish whether the information held by the agents was sufficient to solve the task, this was done via three experiments: the smart traveler, the smart guide, and the unbiased experiment.

[2]The system did not show many variations when using different tuning, hence the same was kept throughout all experiments

Then the environment was altered into a non-static environment (see section 3.5.3.2).

3.5.3.1 Static environment

At first, we considered a "smart" or fully developed traveler (further referred to as the smart *traveler* experiment) and only trained the guide to solve the problem given that the traveler had a set of known instructions it could do pre-determined on only the message sent by the guide. The deterministic mapping is what we refer to as smart in this sense, no training is needed to understand the guide's instructions.

Given the language $\mathcal{L}$ in this setting, there are only four possible messages m, these four messages are, from the traveler's perspective, mapped to one unique action (up, left, down, and right). The deterministic movement of the traveler introduces a bias to the system, which is removed in a later experiment.

Secondly, the reverse was done where the traveler was trained on solving the task given that the guide was "smart" (this is further referred to as the smart *guide* experiment). Since the two agents attempt to convey different forms of information in their communication, a different interpretation of "smart" had to be made; given that the guide was successfully trained in the smart *traveler* experiment, the guide's developed network from that experiment was utilized to simulate a smart guide. Hence, the deterministic movement of the traveler from the previous experiment produces some bias in this system as well. Although in this experiment, the traveler would be allowed to reinterpret the messages and map them to different actions than the deterministic ones, reducing some but not all bias.

To remove the bias, training was made on the system as a whole, starting with two untrained agents (further referred to as the *unbiased* experiment). Hence, both of the agents were reinitialized with no knowledge from prior training.

3.5.3.2 Non-static environment

Keeping the same agent settings as in the *unbiased* experiment, the environment shifted to the non-static version. This increases the complexity of the environment significantly and requires the guide to start comprehending the connection between the target value, and the location of the corresponding obstacles shape and color combinations. With the three variables: color and shape combination, target value, and position, the environment has a total of 96 different combinations.

Additionally, to compare if the system made use of the communication in this environment, the communication was completely removed and an experiment was made without the guide. This became a comparable baseline, similar to the one used in section 3.4.5, to distinguish whether the system was successful or not.

4

Results

This chapter presents the results of the experiments, starting with and emphasizing the better results of the discrete environment and later moving into the less prominent and somewhat inconclusive experiments of the continuous world.

4.1 Grid World

The following three sections will present the results of the experiments in section 3.4. For all the following grid world results, the maximum length of an episode was set to 100 time steps. Additionally, the *entropy* graphs in this chapter, show how certain the agent is of its action, a value closer to zero equals a decision that is certain (low randomness), and the higher the value, the more uncertainty (high randomness).

Clarification: the episode length graphs, in this section, are scaled to an average result of 100 episodes on the x-axis. The graphs are also plotted using a weighted average over an additional 100 values to smooth the curve and make it more readable.

4.1.1 No communication experiment

In the no communication experiment of section 3.4.3, the first run had a hidden size of 10 neurons. With this tuning, the traveler's policy converges at about an average episode length of 23 time steps, as seen in figure 4.1. We see by comparing this value to the expected value of equation 3.3 in section 3.4.3 that this setup does not find an optimal search path. When studying figure 4.2, we see that the entropy of the policy is converging to roughly 0 at about $x = 3\ 200$ (after 320 000 episodes), which shows that the traveler is almost exclusively exploiting the policy that has been learned at that point, and is very certain that its actions are the correct ones to take at each state.

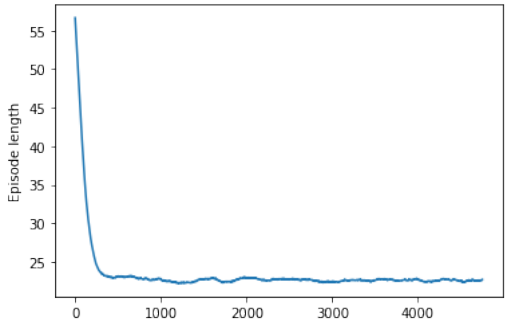

Figure 4.1: The average episode length over 100 episodes, for a total of 500 000 episodes. Using the setup with hidden size 10 in the no communication experiment.

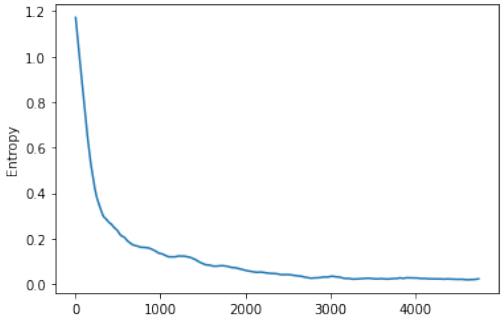

Figure 4.2: The average entropy over 100 episodes, for a total of 500 000 episodes. Using the setup with hidden size 10 in the no communication experiment.

Due to the previous two observations of the converging episode lengths and the exploitation of the policy, the second run had an increased hidden size to a total of 16 neurons in each hidden layer to allow for a more expressive representation of the observations. As seen in figure 4.3 this tuning has significantly better result, the curve starts to flatten out at $x = 4000$ (400 000 episodes), and also the entropy is close to 0 from that point forward (seen in figure 4.4). Hence, when the agent has become certain of its actions, we see that it reaches a final solution at the end of training that takes only about 15 time steps, which is approximately the same and even slightly better than the expected theoretical value of an optimal search path.

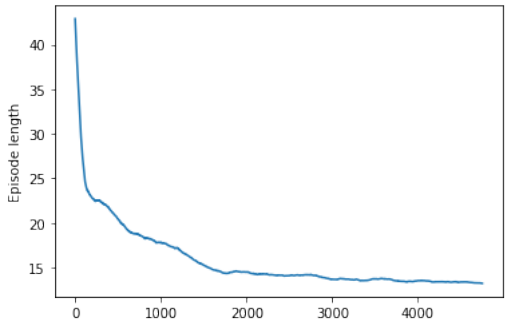

Figure 4.3: The average episode length over 100 episodes, for a total of 500 000 episodes. Using the setup with hidden size 16 in the no communication experiment.

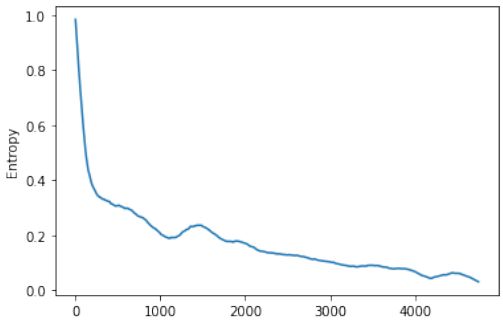

Figure 4.4: The average entropy, for the *traveler*, over 100 episodes for a total of 500 000 episodes. Using the setup with hidden size 16 in the no communication experiment.

4.1.2 One-way communication experiment

The network settings and tuning from the no communication experiment with hidden size 16, were kept for the traveler going into one-way communication experiment.

When adding the communication from the guide the performance increased significantly, as seen in figure 4.5, at about $x = 500$ (50 000 episodes) the system reached a solution that performed equal to the converged policy of the no communication experiment which took about 8 times as many episodes until it reached a solution of such performance. Meaning, both the training time and the final policy could be considered improved for the one-way communication when compared to no communication. At the end of training, the agents achieved a solution to the problem approximately at an average episode length of 4.5. Although, when looking at figure 4.6 we see that the traveler still has some randomness in its policy at later stages of training, and does not fully exploit its current policy. Meanwhile, the guide, who has a more static observation space than the traveler, has converged to almost no randomness in its actions.

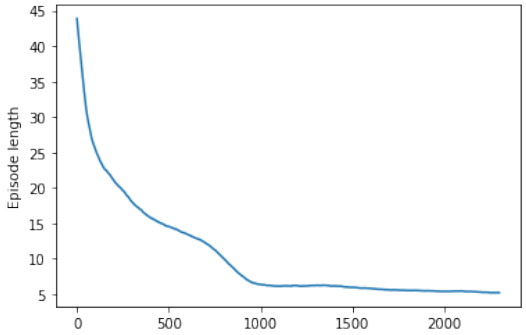

Figure 4.5: The average episode length over 100 episodes, for a total of 250 000 episodes. Using the setup in one-way communication experiments.

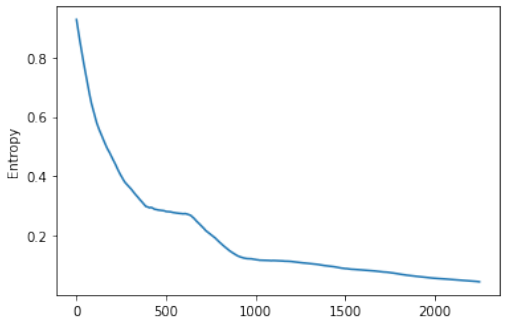

Figure 4.6: The average entropy, for the *traveler*, over 100 episodes, for a total of 250 000 episodes. Using the setup in one-way communication experiments.

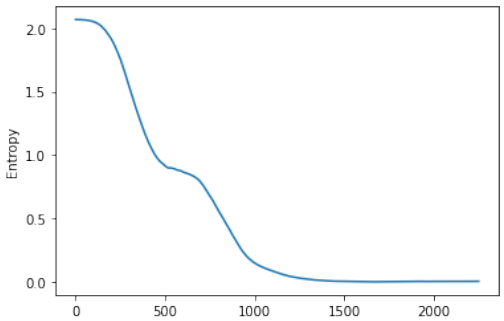

Figure 4.7: The average entropy, for the *guide*, over 100 episodes, for a total of 250 000 episodes. Using the setup in one-way communication experiments.

The guide can make 80 possible observations, since there is only one target (and no distractors), that can spawn at 80 possible positions. In the following two figures 4.8 and 4.9, the color of each tile represents the word that the guide is most likely to send as a message to the traveler, if the target spawned in that tile. Hence, tiles with the same colors are mapped to the same word, and the space is divided into

sectors of the most likely word to be sent. This figure is generated by taking the trained guide of the one-way communication experiment and feeding it every single one of the possible 80 observations. The central white square of the figures is never colored due to the target being unable to spawn at that position. In figure 4.9 the tiles that the traveler can see without moving from its starting position are removed. Since these theoretically need no communication to reach optimally, given that the traveler understands its task. In these figures, we see a very prominent pattern of the agents partitioning the space outside the traveler's initial field of view into different words.

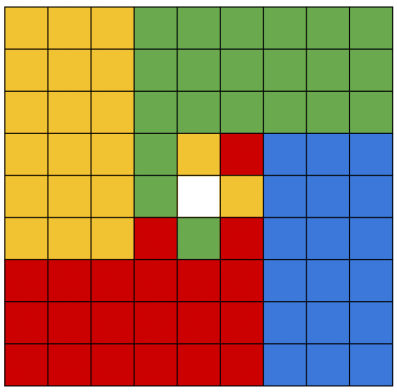

Figure 4.8: The partitioning of the discrete space into sectors, where each color represents the distinct word that the guide is most likely to send given the target has spawned at that tile.

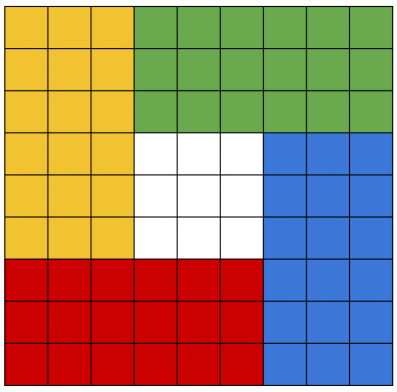

Figure 4.9: The partitioning of the discrete space into sectors, where each color represents the distinct word that the guide is most likely to send given the target has spawned at that tile. Excluding the tiles that the traveler can see directly when spawning.

4.1.3 Two-way communication experiment

As described in section 3.4.5, we trained the system with one-way communication to act as a comparable baseline. We see in figure 4.10 that the system converge to an average episode length of about 13, at $x = 4500$ (450 000 episodes). We also see that the entropy graphs in figures 4.11, and 4.12 quite accurately resembles the ones from the previous one-way communication experiments. Despite the fact that this environment is far more complex, than the previous one.

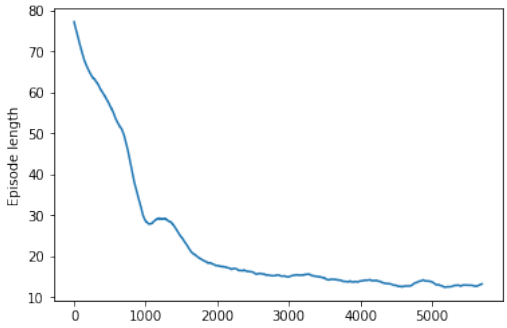

Figure 4.10: The average episode length over 100 episodes, for a total of 600 000 episodes. Using the setup with one-way communication in two-way communication experiments.

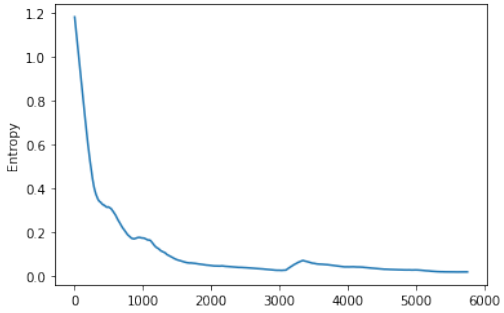

Figure 4.11: The average entropy, for the *traveler*, over 100 episodes, for a total of 600 000 episodes. Using the setup with one-way communication in two-way communication experiments.

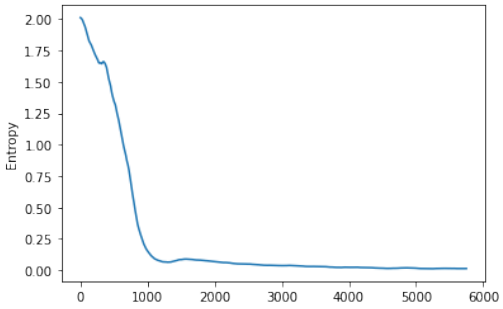

Figure 4.12: The average entropy, for the *guide*, over 100 episodes, for a total of 600 000 episodes. Using the setup with one-way communication in two-way communication experiments.

When studying the two-way communication results, we noticed some completely new characteristics in comparison to what was seen before. We see that the curve in figure 4.13 depicting the episode length is less steep than in previous experiments, meaning that the system converge to an efficient solution later in the training. We also notice the sort of levels of improvement as the training progresses, a repetitive pattern with about every other 100 000 episodes is noticeable, where the first 100 000 episodes contribute to significant improvement and then the second 100 000 episodes have only a slight improvement. This pattern continues throughout the training, with roughly every 100 000 episodes. The final average episode length of the two-way communication is only slightly better than the one-way communication, it converges to an average episode length of 11.

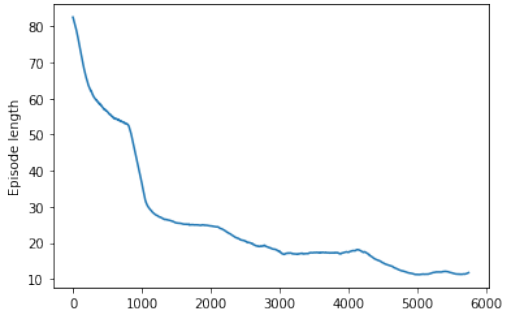

Figure 4.13: The average episode length over 100 episodes, for a total of 600 000 episodes. Using the setup with two-way communication in two-way communication experiments.

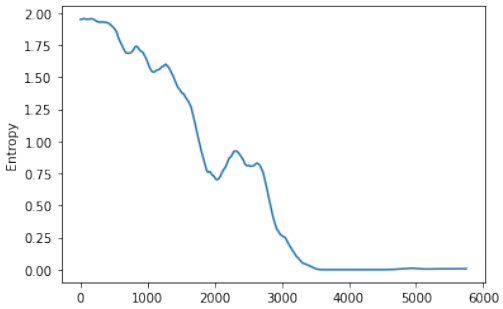

Figure 4.14: The average entropy, for the *traveler's communication*, over 100 episodes, for a total of 600 000 episodes. Using the setup with two-way communication in two-way communication experiments.

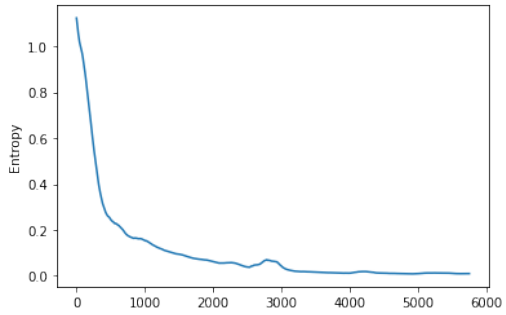

Figure 4.15: The average entropy, for the *traveler*, over 100 episodes, for a total of 600 000 episodes. Using the setup with two-way communication in two-way communication experiments.

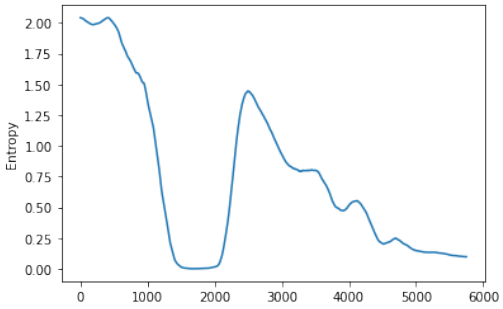

Figure 4.16: The average entropy, for the *guide*, over 100 episodes, for a total of 600 000 episodes. Using the setup with two-way communication in two-way communication experiments.

We also notice a very different characteristic in figure 4.16, where the guide's entropy quickly spikes from almost 0 up to about 1.5, the interpretation of this is that the guide starts "exploring" more of its action space, and is not as certain of its actions anymore. We also notice that this is at about the same time as we see the second improvement in episode length starting to happen, at about $x = 2000$ (200 000 episodes).

When looking at a representation of partitioning of the space, we compared 3 different configurations of walls. For the traveler, we made the traveler visit every available tile and extracted its most probable communication. As seen in figure 4.17, depicting the traveler's most likely message, it is most likely to send the same word no matter where in the environment it is, and no matter which configuration of walls it appears to be in.

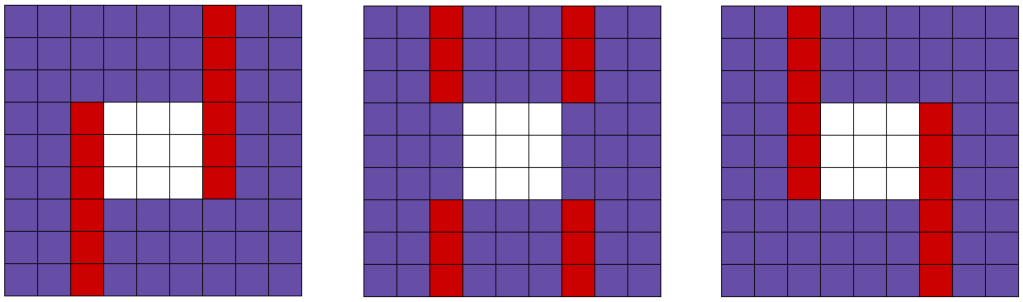

Figure 4.17: The traveler's most probable communication standing at each position in the environment. The red tiles represent the wall configuration of the environment.

The guide's partitioning shows some varying communication, since the traveler seemingly communicates the same message at every tile, this message was used to retrieve the partitioning of the traveler. Every possible target spawn location was generated in the three wall configurations, and the most probable word to be sent was extracted. In figure 4.18 we see the partitioning of the three different configurations. We notice that the area in between the red walls maintains the same partitioning over the different configuration, although the space outside the walls varies from configuration to configuration. Especially the top left corner that switches between yellow and blue.

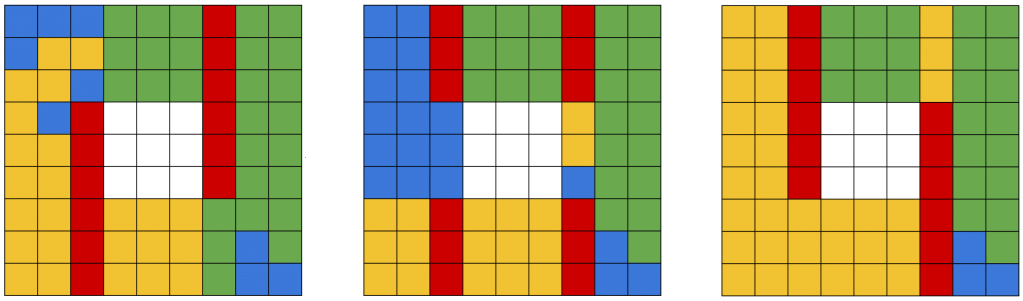

Figure 4.18: The guide's most probable communication standing at each position in the environment. The red tiles represent the wall configuration of the environment.

4.2 Continuous World

Clarification: The graphs in this section show the training progress of the experiments in the continuous world, where the x-axis shows how many decisions have been made (recall that this is done every fifth time step) and the y-axis on shows the episode length scaled one to one of the time step. To find the number of past episodes in the graph, one needs to multiply the x-value by 5 and divide by the average episode length up until that point. Although, since the exact episode of the system is less important when using the PPO trainer, one can simply interpret the x-axis as linear progress of the training.

4.2.1 Continuous world experiments

During the experiments in the static world, it was clear that the system managed to find smart behaviors for both the guide and the traveler. The agents did indeed collectively possess sufficient information to solve the task in the static world, although in the non-static world, the system was not successful in reaching a efficient solution. The following two subsections outline the results of each environment and experiment.

4.2.1.1 Static environment

In the smart *traveler experiment* experiment, which progress is shown by the upper graph in figure 4.19, the guide finds an optimal behavior that allows the traveler to walk straight to the target with each episode (confirmed visually in the simulation), although we see in the figure that this takes about 4 million time steps to achieve.

When switching to the smart *guide* experiments, the system converge to an equally good solution as the smart *traveler* experiments, ≈ 3 times as fast (at about 1.4 million steps). The average episode length, over the progress of the experiment, is shown in the lower graph of figure 4.19. This expediated solution was to be expected since in this test, the guide's "smartness" is a prior mapping to the states of the environment and utilizes a pretrained network, as opposed to in the smart *traveler* experiment where the "smartness" comes from just a deterministic function of the words sent and is not trained by the agents.

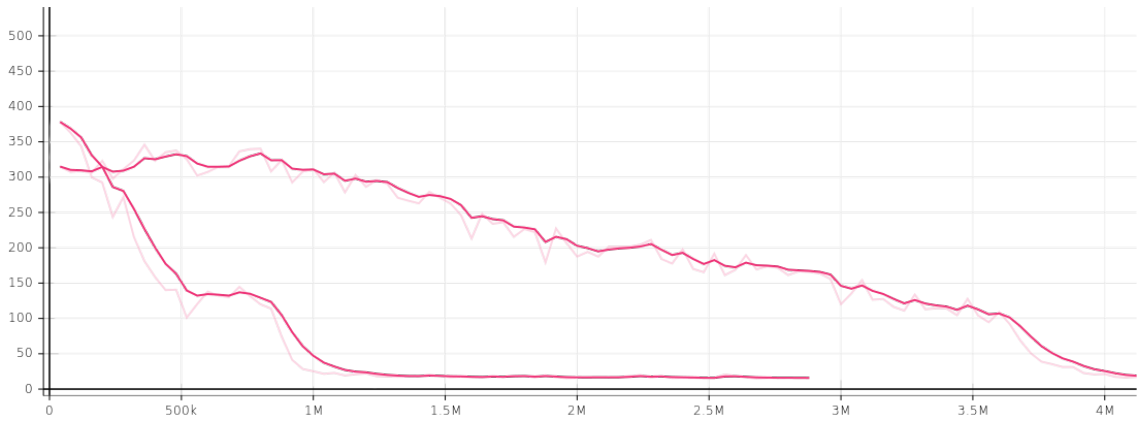

Figure 4.19: Training results in a static world. The upper graph depicts the smart *traveler* experiments, and the lower graph shows the smart *guide* experiments.

In figure 4.20, the results of the *unbiased* experiment, is shown. This training takes about ten times as many steps as the setting with the smart guide, although this is a system without the bias of deterministic movement. This experiment shows that the two agents collectively maintain enough information to solve a static environment in a continuous space.

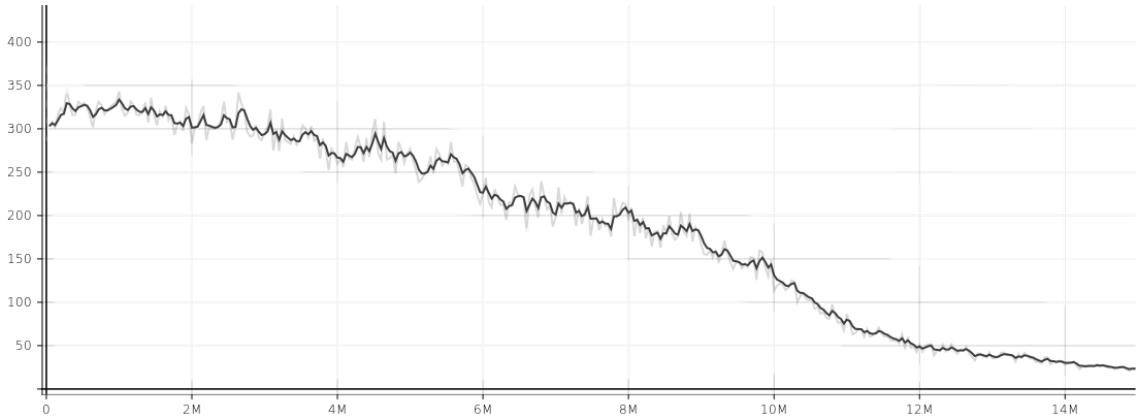

Figure 4.20: Training results from experiments in a static environment. Experiments made with both agents blank, no prior knowledge of the environment nor the task.

4.2.1.2 Non-static environment

When moving into the non-static environment, the results became harder to interpret and less conclusive. Visually, the traveler started acting less certain of the communication provided by the guide. It simply appeared to be guessing, trying to hit all the distractors until it found the correct one. This is also reflected in the graph, where we in figure 4.21 see that both the graphs converge at about an episode length of 200, which is roughly 180 steps longer than the solutions to the static environment.

Both the no communication baseline and the one-way communication converged to a similar result at the end of training. Although, as can be seen in the figure, one-way communication, represented by the orange curve, has a slightly shorter time to reach an average of 300 steps per episode, than the no communication baseline, represented by the blue curve. Whether this is a result of the communication or some other coincidence is too hard to tell from the experiments that have been done, hence this experiment is considered being too inconclusive to show anything useful.

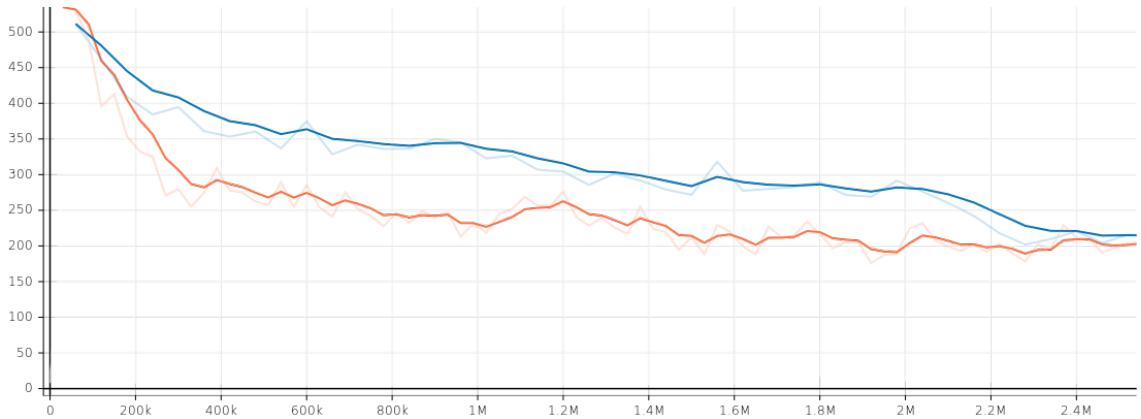

Figure 4.21: Result of the no communication and one-way communication experiments, average episode lengths over time, in a non-static environment

5

Discussion

At first, we aimed to solve the problem in the continuous space, as seen this proved to be far too difficult. In the first approach where we tackled the static continuous environment (found in section 3.5), ultimately the only main goal became to establish whether the traveler could understand the guide's instructions or not. As seen in the results, this proved true, although this environment gave little to no results as a system that could attempt to solve a non-static version of the world.

A seemingly random guessing behavior of the traveler appeared as soon as we moved into the non-static environment, this might have been a result of the reward signal being tuned and/or designed wrongly. We speculate that, if the punishments were large (about the same magnitude as the positive reward) then the traveler would avoid interacting with the obstacles, and if the punishment were smaller (in the range of 2-5 times smaller than the positive reward) the behavior seemed increasingly random, relative to the difference between the two values. We interpret this behavior to stem from the fact that the lower punishment gives to low incentive to avoid them. Finding the balance in the relation between reward and punishment was a difficult challenge, with this design. This also drove a redesign of this relation in the grid world, which gave the reward signal mentioned in section 3.4.[1]

Due to the difficulties in the continuous environment, we moved to the discrete environment about halfway through the work, since continuous movement made the problem unnecessarily difficult to solve. As we have shown, the results of this world were much more conclusive and helped show that the agents are able to solve the problem and develop a mutual language. At this point, we were also more considerate of the complexity of the environment when starting to develop it, and then increased the complexity of the environment as we got some results. Although, this forced us to constrict the research more and not reach what was the initial idea, hence there are no two-way communication results in the continuous world and less effort was put into attempting to solve that world, shifting focus into the grid world instead. We consider this to be the correct choice due to the time span of the work and the difficulty of the continuous world.

[1] As a side-note: In some additional tests done in the continuous space, that are not included in this work, the reward signal of the grid world did not yield any better results than previously seen in the continuous world.

5.1 Language

The language size was something that we discussed a lot at the start of the project, but as soon as we started to get some results in the grid world, we noticed, in the experiments that were successful, that as long as there were at least 4 words in the language, system was able to achieve a result near-optimal. We believe that this number is only for the given environments that were used in this work, and due to their small size and simple complexity. This is coherent to the results of M.Kågebäck et al. [11], where they use a language vector of size 50 and the agents partitioned the color space into fewer words than the size of the vector.

The results in section 4.1.2 show that the agents omit the unnecessary words when making an optimal solution since the number of sectors (4) is less than the size of the language (8). We know from the theoretical optimal search path in section 3.4.4 that dividing the environment into 4 sectors would be enough for this environment, such that the traveler can choose an optimal path toward the target.

However, if the guide had chosen to use the whole language (8 words), the same final average episode length had been reached. Assume that the target spawns somewhere in the upper right corner (dark green sector in figure 5.1). One optimal path toward this sector is shown in figure 5.2, when taking this path, the traveler also passes the light green sector. Therefore, you can merge these two sectors into one sector (seen in figure 4.9), which implies using a smaller language. We also believe that discarding the unnecessary words shortens the training time until convergence.

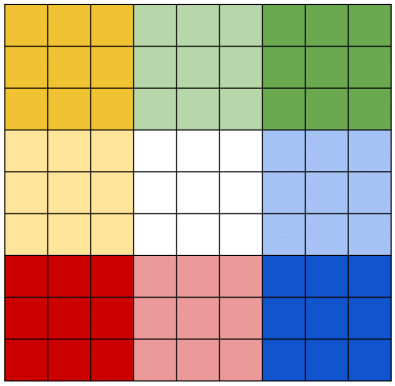

Figure 5.1: A 9×9 grid world, divided into 8 sectors.

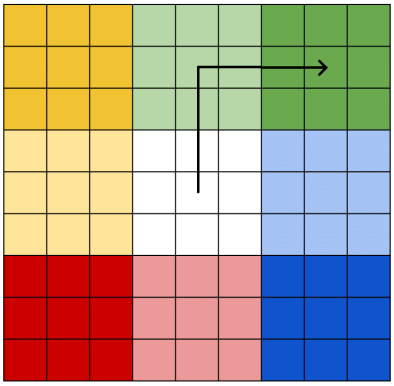

Figure 5.2: A 9×9 grid world, divided into 8 sectors. The arrow represent an optimal path towards the dark green sector, starting from the midpoint.

When comparing figures 4.8 and 4.9 we see that it is only the squares adjacent to the traveler's starting position that do not have a prominent pattern. This is most likely since there is no need to listen to the guide's information once the traveler sees the target and knows it is looking for it.

It is way more difficult to interpret any patterns in the resulting partitioning of

the space in two-way communication, seen in figures 4.17 and 4.18, some sectors maintain their partitioning over configurations, e.g. the space in between the walls, the general disparity between the upper right and the lower left, as well as the lower right corner. It's shown in [14], that when removing one of the agents messages from the observation, it does not significantly reduce the performance. When looking in figure 4.17, we can see such a pattern, the traveler learns to send the same message, which is equivalent to not sending any message. We speculate that the guide has a behavior of negative listening [14], it discards the traveler's message, regardless of whether the signaling is positive or not.

Although we see in the results of the experiment, the system reaches at least an equal result as the one-way communication, meaning that this partitioning, that seems less prominent to us, still allows the agents to solve the task efficiently. Whether this stems from another relation than the partitioning of the space or not, needs to be evaluated further in order to establish some concrete proofs.

5.2 Two-way communication

A simple environment, made the problem much easier to solve. However, with two-way communication, it did not make much sense to have a simple environment. The traveler would not have enough information to convey to the guide. That is why the trade-off between a simple vs complex environment became highly important and a big challenge for us. We wanted an environment as complex as possible such that the traveler had more things to convey, while still keeping it simple enough to be solvable for the MAS. It seems unclear whether two-way communication gives any significant benefit in this size and complexity of an environment. The final average episode lengths are simply too close to each other to tell, even though the two-way communication converges to about 2 time steps less than the one-way communication.

On the other hand, the behavior mentioned in section 4.1.3 where the guide seemingly leaves its local optima and then finds a more optimal solution, at this point we speculate that the guide explored a new semantic meaning of the word(s) that improved the agent's performance. Also, the entropy spike entails that the guide starts exploring the states more than the previous episodes (i.e. when the entropy is at zero), leaving a local optima. What is interesting about the spike is the discrepancy between one-way communication, where such a spike is nowhere to be found, and two-way communication, where it helps the system find a better optima.

We theorize that this behavior might have two explanations. Firstly and foremost, the traveler's communication with the guide, since this gives the guide some dynamic information with each step. This allows the guide to end up in some different state than it thought it expected, which eventually proves to be a better solution. Secondly, the system could be sensitive to hyperparameter tuning and a slight change to the learning rate could make the policy "step out" of a local optima every time, to find the better global optima. Although the latter explanation seems less likely to, on its own, enable such behavior, since the learning rate remained the same for

the one-way and two-way communication in this environment.

Hence, we think that this communication setting could possibly be more beneficial in environments with bumpier loss landscapes, than the ones used here.

5.3 Algorithms and Policy Gradient Optimization

The two kinds of trainers that have been used in the work: PPO and REINFORCE with a baseline augmentation, are algorithms designed with single-agent systems in mind, and were hence used with decentralized training and execution. This choice was motivated by the scale of the work and the low system requirements to train/evaluate an experiment.

For future work, it would be a sound idea to compare these algorithms to others that are designed specifically for MASs, especially algorithms with centralized training in mind since they can establish a global hierarchy for the entire MAS during training. For example, Reinforced Inter-Agent Learning (RAIL) and Differentiable Inter-Agent Learning (DIAL) proposed and used by J.Forester et al. [7], or Multi-Agent POsthumous Credit Assignment (MA-POCA) explained by A. Cohen et al. [5], which in turn, is also a multi-agent trainer that can be used in the Unity ML-Agents toolkit.

6

Conclusion

It is shown throughout the work that the agents can indeed collaborate and solve the navigation task efficiently by developing a language that had no prior meaning in a discrete setting. We see that the space of the discrete world becomes partitioned into sectors represented by distinct words. The sectors resemble an optimal search path, in the cases where we have been able to establish/find such a path.

We can also conclude that in a continuous world the collaboration becomes much more difficult and different models and environments than the ones used in this work need to be experimented with to see the benefits in such a setting.

Additionally, we also found that the two-way communication was not expediting the training time of the grid world, but rather extending it. It is also unclear whether it changes the outcome in any significant way, given that the system is allowed to train for an unlimited time. Although, as discussed, this might be because the environments used to evaluate this method were simply not complex nor big enough to benefit from two-way communication, and hence the method should not be dismissed in the future but rather elaborated upon in a larger and more complex environment as it did show some interesting behaviors.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: A system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[3] Richard Bellman. A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.

[4] Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. *arXiv preprint arXiv:1804.03980*, 2018.

[5] Andrew Cohen, Ervin Teng, Vincent-Pierre Berges, Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, and Sujoy Ganguly. On the use and misuse of absorbing states in multi-agent reinforcement learning. *arXiv preprint arXiv:2111.05992*, 2021.

[6] Gautier Dagan, Dieuwke Hupkes, and Elia Bruni. Co-evolution of language and agents in referential games. *CoRR*, abs/2001.03361, 2020.

[7] Jakob N Foerster, Yannis M Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676*, 2016.

[8] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

[9] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[10] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

[11] Mikael Kågebäck, Emil Carlsson, Devdatt Dubhashi, and Asad Sayeed. A reinforcement-learning approach to efficient communication. *Plos one*, 15(7):e0234894, 2020.

[12] Ivana Kajić, Eser Aygün, and Doina Precup. Learning to cooperate: Emergent communication in multi-agent navigation. *arXiv preprint arXiv:2004.01097*, 2020.

[13] Angeliki Lazaridou and Marco Baroni. Emergent multi-agent communication in the deep learning era. *arXiv preprint arXiv:2006.02419*, 2020.

[14] Ryan Lowe, Jakob Foerster, Y-Lan Boureau, Joelle Pineau, and Yann Dauphin. On the pitfalls of measuring emergent communication. *arXiv preprint arXiv:1903.05168*, 2019.

[15] William P McCarthy, Robert D Hawkins, Haoliang Wang, Cameron Holdaway, and Judith E Fan. Learning to communicate about shared procedural abstractions. *arXiv preprint arXiv:2107.00077*, 2021.

[16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[17] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[21] Ronald J. Williams. Simple statistical gradient-following algorithms for con-

nectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992.

[22] Karl Johan Åström. Optimal control of markov processes with incomplete state information i. 10:174–205, 1965.