UNIVERSITY OF PATRAS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AUTOMATED AND CONTROL SYSTEMS

Diploma Thesis

Multi Agent Exploration with Reinforcement Learning

Alkis Sygkounas

AM:1056170

Supervisors:

Demosthenes Kazakos, Assistant Professor (University of Patras)
Dimitris Tsipianitis, Instructor (University of Patras)
George Nikolakopoulos, Professor (University of Lulea)

Committee Members:

Demosthenes Kazakos, Assistant Professor
Bechlioulis Charalampos, Associate Professor
Dimitris Tsipianitis, Instructor

Mentors:

Samuel Karlsson
Sina Sharif Mansouri

Patras, September, 2021

Declaration

Acknowledgments

I would like to express my special thanks of gratitude to my supervisors Samuel Karlsson, Sina Sharif Mansouri as well as the Head of Subject George Nikolakopoulos and the Professor and Examiner Dimitris Tsipianitis, who gave me this opportunity to do this wonderful project on the topic (Multi agent exploration using Reinforcement Learning), which also helped me in doing a lot of research and i came to know about so many new things I am really thankful to them.

Secondly i would also like to thank my parents, who helped completing this diploma thesis because without them this could not be possible.

Patra, September/2021

Contents

List of Figures

List of Tables

Abstract

Modern mobile robots have begun to be used in many exploration and exploration and rescue applications. They are essentially coordinated by human operators and work with inspection or rescue teams. Over time, robots (agents) have become more sophisticated and we see more autonomy in more complex environments. Therefore, the purpose of this diploma thesis is to present an approach for autonomous multi-agent coordination for exploring and covering unknown environments. The method we suggest is reinforcement learning in combination with the use of neural networks (Deep Learning) to plan the course for each agent separately and also the effort to achieve collaborative behavior between them. Specifically, we have used two techniques applied in recent years, which are the target neural network and the prioritized experience replay, which have been proven to stabilize and accelerate the training process. Also, agents must avoid obstacles (walls) throughout the exploration. We have also solved the problem of the need for prior information / knowledge about the environment, thus using only the local information available at any given time to make the decision of each agent. Also, in addition to the route created for exploring the maps, agents successfully avoided obstacles (walls). In this diploma thesis, the exploration of the unknown environment is done in a two-dimensional model (2D) using multiple agent for a variety of different maps, ranging from small sizes to and large sizes. Also, the map that our model is checked will be either randomly made or in a specific model of a real cave. Finally, the efficiency of the exploration is investigated for a different number of agents and for a different type of neural network.

Chapter 1

Introduction

Autonomous flight for UAVs (unmanned aerial vehicles) has become an increasingly important research area in recent years. One of the major challenges with autonomous motion planning is ensuring that an agent can efficiently explore a space while avoiding collision with objects in complex and dynamic environments. To resolve this, recent research has turned to applying deep reinforcement learning techniques to robotics and UAVs. For the last years, the number of applications where mobile robots collaborate with humans [1] dramatically increased. Mainly, this is due to advances in batteries, powerful computation boards, and miniaturization of sensors. Recent advances in unmanned aerial vehicles (UAV) have allowed mapping and exploration in difficult to access areas that were previously not possible using unmanned ground vehicles. UAVs have been deployed in areas that are deemed dangerous for human operation, and provide important information about the environment in applications such as search and rescue, site inspection, victim search in disaster situations and monitoring. UAV must be designed to operate autonomously with no prior information about the environment Moreover, they are used in disaster search and rescue missions [2], utility inspection [3] crop monitoring [4], etc. Depending on mission requirements, mobile robots could be equipped with different types of sensors or actuators that depend on the mission objectives. Their assistance in inspection, search, and rescue missions, and the exploration of unknown areas plays crucial role and provides benefits to human operator safety, decision making, fast exploration, three-dimensional (3D) reconstruction of the environment and providing localization.

In this diploma thesis we introduce a reinforcement learning approach (RL) [5] of exploring and navigating in different and unknown maps with different amount of agents without any previous knowledge of the area. Specifically, we tried to cover as much area as possible with multiple agents, while at the same time we avoided obstacles(walls). We used a variation of Q-learning [6] . We also utilized the prioritized experience replay [7] and also we tested different neural structures [8] for our problem. The maps are two dimensions with size of grid by grid. The testing (and training) were concluded in different size of maps with different number of randomly generated each time walls. Also, different

number of agents were tested for determining the coverage percentage of the maps. The resolution of each grid cell is not taken in consideration in our approach not either the limited battery operating time of the agents in a real life scenario.

The master thesis is structured, as follows. Initially, the is presented in Section 1 while the background is presented in Section 2 and also our proposed method for the multi-agent exploration is established in Section 3. The tests and the results are presented in Section 4. In Section 5 we discuss the results and finally we conclude the master thesis in Section 6. Also, we will refer any type of robot(UAV,ground robot etc) as an agent for the rest of the master thesis.

Related works/Contribution

Many multiple robot area coverage algorithms have been developed over the past decades in the robotics literature. [9] provides a survey of area coverage algorithms using single and multiple robots. [10] propose a multi-robot algorithm for effective coverage by decomposing the area into cells and allocating cells to robots for covering. [11] present a polynomial time spanning tree algorithm for a single robot to cover an area. [12] present coverage algorithms in which the robots spread out and move away from each other while covering the complete area. More present day works are using Deep Learning [13] with many different techniques like the Delayed-Learning trick [14] for achieving the respected goal. Also we see a rise in the usage of Reinforcement Learning approach for exploring or controlling agents in unknown and uncertain environments [15].

Our contribution is focused in the multiple agent exploration and path planning with cooperative coverage of an unknown area in the presence of obstacles using only local information (observations). Specifically, we trained and tested the agents on unique two dimension maps (2D) with the Reinforcement Learning approach .We also used the prioritized experience replay [7] and the target neural network [16], for achieving better performance and stability in our model. Furthermore, the decision-making of each agent is based on the Surrounding information of their local environment and they do not have any prior knowledge of it. Also, for achieving cooperative coverage and general a satisfying behaviour, it was necessary for this work, that the agents were sharing the gps-coordinates. Finally, we tested our model for different 50x50 (random maps) and 100x50 (based on a real cave map) and for different number of agents and different neural net structure. We show the paths for each agent that they took and also graphs for total (median) coverage for a large number of episodes.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning(RL) [17] is learning what to do, how to map situations to actions, so as to maximize a numerical reward. The agent is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Thus the agent is rewarded or punished based on the actions he took and results he made.

In the standard RL model an agent is connected to its environment via perception and action. On each step of interaction the agent receives an input, some indication of the current state of the environment and then the agent chooses an action a to generate as output. The action changes the state of the environment and that generates a reward which is a scalar reinforcement signal. The agent's behaviour should choose actions that increase the long-term sum of values of the reinforcement signal.

2.2 Markov Decision Processes(MDP)

A mathematical representation of a complex decision making process is "Markov Decision Processes" (MDP) [18] . MDP is defined by:

• A state S, which represents every state that one could be in, within a defined world.

• A model or transition function T; which is a function of the current state, the action taken and the state where we end up. This transition produces a certain probability of ending up in state S', starting from the state S and taking the action A.

• Actions are things I can do in a particular state.

• A reward is a scaler value for being in a state. It tells us the usefulness of entering the state.

The final goal of the MDP is to find a policy that can tell us, for any state, which action to take. The optimal policy is the one that maximizes the long-term expected reward. There are two properties on which the Markovian process is based on: Only the present

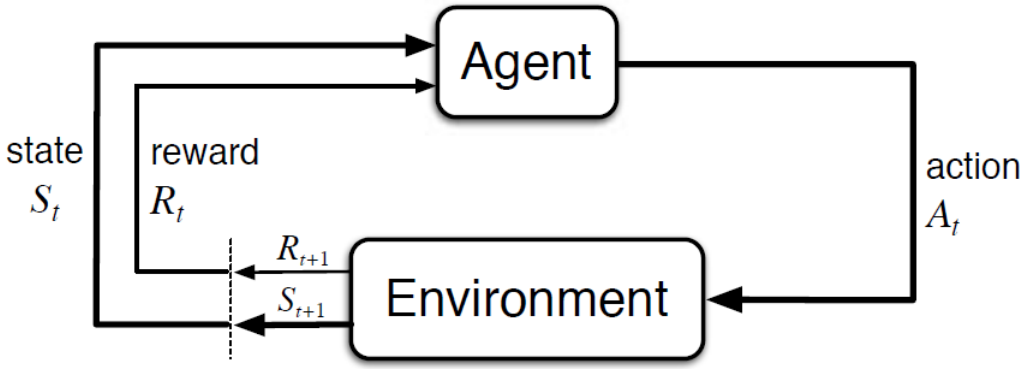


Figure 2.1: Agent-environment interaction in MDP

matters, which means that the transition function only depends on the current state S and not any of the previous states.

2.3 Exploration vs Exploitation

On-line decision making involves a fundamental choice; exploration, where we gather more information that might lead us to better decisions in the future or exploitation, where we make the best decision given current information. This comes up because we're learning on-line. In the reinforcement learning setting, no one gives us some batch of data like in supervised learning. We're gathering data as we go, and the actions that we take affects the data that we see, and so sometimes it's worth to take different actions to get new data.

Exploration allows an agent to improve its current knowledge about each action, hopefully leading to long-term benefit. Improving the accuracy of the estimated action-values, enables an agent to make more informed decisions in the future.

Exploitation on the other hand, chooses the greedy action to get the most reward by exploiting the agent's current action-value estimates. But by being greedy with respect to action-value estimates, may not actually get the most reward and lead to sub-optimal behaviour. When an agent explores, it gets more accurate estimates of action-values and when it exploits, it might get more reward. It cannot, however, choose to do both simultaneously, which is also called the exploration-exploitation dilemma [19]. A solution to that is the epsilon greedy policy and the selection of action in each time step t is shown in the figure 2.2.

2.4 Model-Based and Model-Free policy

In a model-based RL environment, the policy is based on the use of a machine learning model. To better understand RL Environments/Systems, what defines the system is the policy network. Knowing fully well that the policy is an algorithm that decides the action

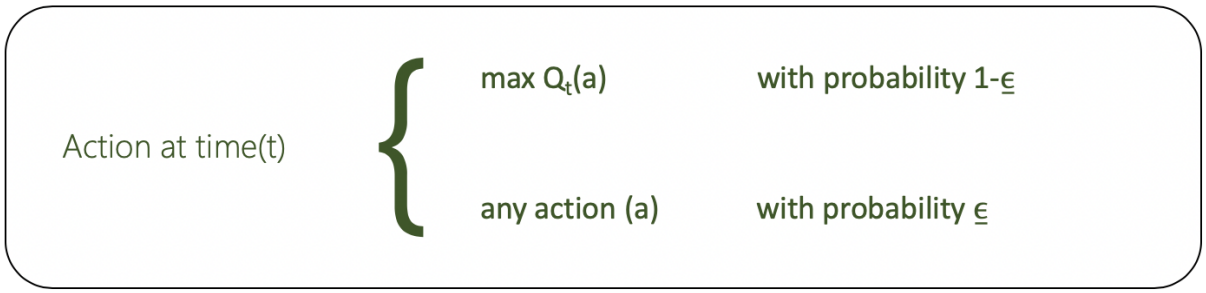


Figure 2.2: Epsilon greedy policy

of an agent. In this case, when an RL environment or system utilizes the use of machine learning models like random forest, gradient boost, neural networks, and others, such an RL system is model-based. Moreover, this isn't the case with a model-free RL, such a system has no policy based on the use of machine learning models; its policy is guided by the use of non-ML algorithms.

2.5 Multi agent Reinforcement Learning

In a fully observable environment in each time step t, each agent observes the current state S_t and chooses an action a according to a policy π.Every action of each agent will form a joint action U_t and all agents will receive a reward $R_t = r(S_t, U_t)$ and the environment will transit to a new state S_{t+1}. On our problem we consider every agent as independent and every each of them has his own neural net(independent Q-learning). In the IQL approach each agent considers the other agents as part of the environment.In each time step t, each agent observes the state S and performs an action U and all agents form a joint action U_t. From the joint action U_t, all agents receive a reward R_t and the environment transits to the new state S_{t+1}.

2.6 Q-LEARNING

Q-learning [20] is a form of model-free reinforcement learning. It can also be viewed as a method of asynchronous dynamic programming (DP). It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains Given an action u_t that generates a reward r_t for the state s_t and observing a new state s_{t+1}. The Q-function is updates according:

$$Q_{t+1}(s_t, u_t) = Q_t(s_t, u_t) + a(r_t + \gamma max_{t+1} Q_{t+1}(s_{t+1}, u_{t+1}) - Q(s_t, u_t)) \qquad (2.1)$$

where a is the learning rate with a value between $(0, 1]$ and γ is the discount factor that determines the future rewards.

2.7 Deep Q-LEARNING(DQN)

For most problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a. Instead, we train a function approximator, such as a neural network with parameters θ, to estimate the Q-values, i.e $Q(s, a; \theta) \approx Q(s, a)$. This can be done by minimizing the following loss at each step i:
$L_i(\theta_i) = E_{s,a,r,s'}[(y_i - Q(s, a; \theta_i))]^2$,where $y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$.
Here, y_i is called the TD (temporal difference) target and $y_i - Q$ is called the TD error. Note that the parameters from the previous iteration are fixed and not updated. In practice we use a snapshot of the network parameters from a few iterations ago instead of the last iteration. This copy is called the target network. Q-Learning is an off-policy algorithm that learns about the greedy policy $a = \max_a Q(s, a; \theta)$, while using a different behaviour policy for acting in the environment/collecting data. This behaviour policy is usually a greedy policy that selects the greedy action with probability $1 - \epsilon$ and a random action with probability ϵ to ensure good coverage of the state-action space.

2.8 Loss function and Optimizing methods

A deep learning neural network learns to map a set of inputs to a set of outputs from training data. Typically, a neural network model is trained using the stochastic gradient descent optimization algorithm and weights are updated using the backpropagation [21] of error algorithm.

In the context of an optimization algorithm, the function used to evaluate a candidate solution (i.e. a set of weights) is referred to as the objective function. We may seek to maximize or minimize the objective function, meaning that we are searching for a candidate solution that has the highest or lowest score respectively. Typically, with neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply “loss.”

The “gradient” in gradient descent refers to an error gradient. The model with a given set of weights is used to make predictions and the error for those predictions is calculated. The gradient descent algorithm seeks to change the weights so that the next evaluation reduces the error, meaning the optimization algorithm is navigating down the gradient (or slope) of error. How you should change your weights or learning rates of your neural network to reduce the losses is defined by the optimizers you use. Optimization algorithms or strategies are responsible for reducing the losses and to provide the most accurate results possible.

2.9 Double Deep Q-Learning(DDQN)

The 2.1(Bellman equation) provides us with the value of $Q(s, a)$ via $Q(s', a')$. However, both the states s and s' have only one step between them. This makes them very similar, and it's very hard for a Neural Network to distinguish between them. When we perform an update of our Neural Networks' parameters to make $Q(s, a)$ closer to the desired result, we can indirectly alter the value produced for $Q(s', a')$and other states nearby. This can make our training very unstable. To make training more stable, there is a trick, called target network, by which we keep a copy of our neural network and use it for the $Q(s', a')$ value in the Bellman equation [6]. Specifically we train the neural net based on:

$$Q(s, a, \theta_{main}) = r + \gamma max_a Q(s_{t+1}, a_{t+1}, \theta_{target}) \tag{2.2}$$

We update the weights of the target network equal to the weights of the weights of the main model every c-iterations. The θ_{main} and the θ_{target} are the weights of the main and target networks.

2.10 Dueling neural networks

The key insight to this architecture [8]is that for many states we don't need to estimate the value of each action choice. In some states tho, is paramount to know which action (like avoiding actions that lead to a collision), but in many other states the choice of action has none repercussion on what happens. For that purpose, a single neural net is used to estimate two separate functions, the value function V that measures how good is to be in a particular state and an advantage function A that measures the relative importance of choosing a particular action based on a particular state. These two functions are calculated separately, each one in its own "stream" and then they being aggregated to calculate the Q-value. The dueling module is described as followed:

$$Q(s, a;\theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{2.3}$$

The parameters α, β are the parameters of the 2 streams of the fully connected layers and θ are the weights parameters of the common layers. Equation 2.3 is unidentifiable in the sense that given Q we cannot recover V and A uniquely. To see this, add a constant to $V(s; \theta, \beta)$ and subtract the same constant from $A(s, a; \theta, \alpha)$. This constant cancels out resulting in the same Q value. This lack of identifiability is mirrored by poor practical performance when this equation is used directly. To address this issue of identifiability, we can force the advantage function estimator to have zero advantage at the chosen action. That is, we let the last module of the network implement the forward mapping:

$$Q(s, a;\theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) - \max_{a' \epsilon |A|} A(s, a'; \theta, \alpha) \tag{2.4}$$

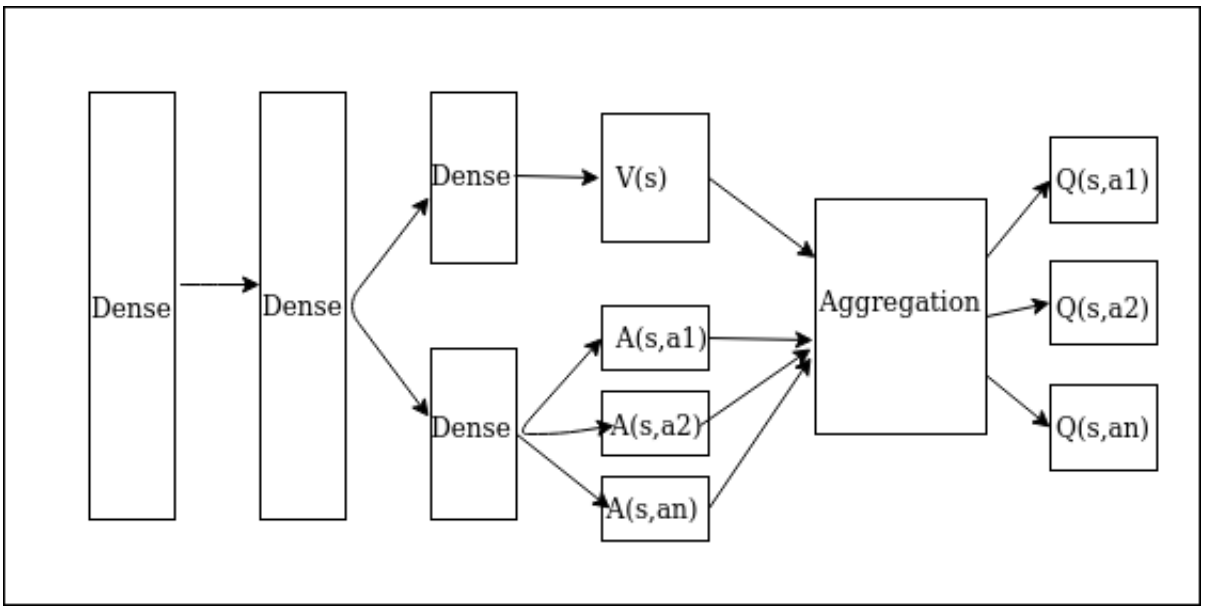


Figure 2.3: Dueling Neural Net

Another module that replaces the max operator with an average is shown as follows:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) - \frac{1}{A}\sum_{a'} A(s,a',\theta,\alpha) \qquad (2.5)$$

Since both equations gave similar results, we used the equation 2.5 for the experiments.

2.11 Prioritized experience replay(PER)

In the classic DQN approach the agent takes experiences uniformly at random that are stored in his memory in order to train. So, all experiences are treated equally and directly being in contrast to the real life experience in which some experiences will be more beneficial than others. To accomplish that, we use the PER method sample experiences to a probability p_t is proportional to the absolute difference between target and estimation values. That being said the p_t is given by:

$$p_t \sim (|r_t + \gamma max_{t+1} Q(s_{t+1}, u_{t+1}; \theta') - Q(s_t, u_t; \theta)| + \epsilon)^{\omega} \qquad (2.6)$$

where ϵ is a small positive constant ensuring that there is no experience with zero probability of being sampled. Also, ω determines how much prioritization is being used with $\omega = 0$ being the uniform case.

Chapter 3

The proposed RL method

3.1 The RL algorithm

For the training phase our algorithm works like the figure 3.1. Specifically, we initialize a memory buffer of a standard size ,in which we store our current state,the next state, action and reward for each step in each episode. Then, each time-step the agents take an action based on some policy and observe the reward and the new state. After a specific number of replay steps , we train the agents based on the store experiences using the Huber loss function. For choosing the previous experiences we used prioritized experienced replay since it performs better than uniform experience replay. Also, in all cases we used the double neural technique(DDQN). In the image 3.1 represent our main algorithm that we use for our problem.

Algorithm 3.1 DDQN algorithm

Initialize Q(s,a) arbitrarily

Initialize memory buffer

for *each episode repeat* **do**

  Initialize initial state

  for *each step in episode* **do**

    Choose a from s using policy derived from Q(e.q , e-greedy)

    Take action **a** and observe **r,s'**

    Store in memory buffer **s,s',a** and **r**.

    Calculate **Q** from target net.

    Every r-replay steps train the agent based on previous experiences

    Evert c-steps update target network equal to main network

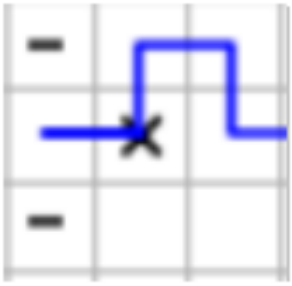
    Set **s'=s**

  end

end

3.2 Local information and input state

Due to lack of any previous knowledge of the maps , the agents dont know the size of the map. So, the only information that is available is the local observation of each agent from his current location. Specifically, each cell on the map can have 3 possible states, meaning that it is either unexplored or already explored or it has a wall. In our case the agents have 1-cell in each direction(and diagonally) in depthas sensed input. In other words they have a square like of local information from the current position. So the input state is a total number of 9(8+information of the current position cell) plus the gps coordinates of the agents, as we want to share basic information between them. Specifically each agent has the gps-coordinates information of the one that is closest to him plus his own. So the total input size is 13. In the figure 3.1 we can see how the agent proceeds his local information. With **-1** we represent the undiscovered squares, with **0** the already discovered squares and with **1** the occupancy of a wall. The x represents the



1	0	0
0	0	0
1	-1	-1

Figure 3.1: Local observations

current position of the agent and the blue line the path he has done already.

3.3 Neural net structure

We considered two different neural net architectures for our experiments . The first one is the most common of all the feed forward neural net. The second architecture is the dueling neural network. In our first model, our net is consisted of the input layer which contains the input state(local information plus gps-coordinates). Next, we have three hidden layers , each one containing 512 nodes and the output layer which contains the actions that the agents can make and those are: up, down ,left,right and stay. So the output size of the neural net is 5 with each one output representing the different actions, that the agent can make. In the second model, again we have the same input size of 13, then for calculating the $V(s)$ we have 3 separate hidden layers and again another 3 separate layers for the $A(s)$. We combine(aggregate) both of them in order to get the output of our neural net which contains the available actions of our agent and again the size of it is again 5.

3.4 Loss function and optimizer used in the proposed method

The Huber loss is a loss function used in robust regression, that is less sensitive to outliers in data than the squared error loss. This function is quadratic for small values of a and linear for large values, with equal values and slopes of the different sections at the two points where $|a| = \delta$. Huber loss is both mean squared error (MSE) and mean absolute error(MAE) means it is quadratic(MSE) when the error is small else MAE. Here delta is the hyper-parameter to define the range for MAE and MSE which can be iterative to make sure the correct delta value. From the equation we find that when the error is less than delta, the error is quadratic else it is absolute. The huber loss function is shown in the figure 3.2. We used Huber Loss function for the rest of the article.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \leq \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Figure 3.2: Huber loss function

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$
$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : *Initial Learning rate*
ν_t : *Exponential Average of squares of gradients*
g_t : *Gradient at time t along* ω^j

Figure 3.3: RmsProp optimizer

Gradient descent is probably the most popular and widely used out of all optimizers. It is a simple and effective method to find the optimum values for the neural network. The objective of all optimizers is to reach the global minimum, where the cost function attains the least possible value. Each time we find the gradient and update the values of weights and biases, we move closer to the optimum value.

RMSprop is a gradient-based optimization technique used in training neural networks. Rmsprop was developed as a stochastic technique for mini-batch learning. RMSprop uses a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding and increasing the step for small gradients to avoid vanishing. Simply put, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyper-parameter. This means that the learning rate changes over time. The RmsProp update rule is being shown on the figure 3.3. We used the RmsProp for the rest of the article, because we saw that it work the better than adam in general.

3.5 Reward function

For the reward function we want to award the agent for discovering new spots but we don't want to punish if the agent goes to a previously discovered spot , as they may need to pass from it or come back from a dead end(like the maze maps in 4.3. So we give 2 points for each newly discovered place and we punish -10 for each movement that results to a wall. Also, we punish -1 for each time-step in order to try to maximize the newly discovered place in correlation to time.
For each time step (t) the reward R_t is given in the equation 3.1.

$$R_t = 2 * new_{discovered} - 10 * wall_{hit} - 1 \tag{3.1}$$

Equation 1: Reward function

3.6 Replay steps

In each time step we have the state s_t, the action a the new state s_{t+1} and the reward r. Each time we store these in formation to the memory buffer and if the current step has a multiple value of the replay steps, then we train all the agents. That means that if the max time-step is 1000 and the replay steps are 2 that means that in an episode the agents will be trained 500 times.

3.7 Training phase

In order to achieve good behaviour we tried different optimizers, different learning rates, different number of epochs etc. Because the investigation of all the different values of the hyper-parameters is impossible, we are showing on the 4 the values that we used.

Moreover the training phase is divided in parts. The initial training phase which contains the exploration part(with epsilon probability), in which the agent learned to avoid the obstacles and explore the map a bit. If the agents had good behaviour meaning that they didn't collide to any wall and had an at least minimum exploration(20-25%) exploration, then we increased the number of epochs(otherwise we kept it to 1). For the optimizing phase, we decreased the learning rate, the total number of epochs (if exploration was good) and increased the value of the replay steps. Also we zero the epsilon probability(exploration) and we increased the beta value(per memory model) to 1. Lastly we increased the batch size as we saw that it improved our model's behaviour .

Chapter 4

Tests Results

The tests on the maps were done in python created environment. We tested on different size of maps the behaviour of the RL algorithm that is shown in figure 3.1 for the different neural nets structures. Also, we tested the total coverage% ratio for different number of agents. The maps are being randomly generated with the % of walls that are occupying the maps being between 15-20% of its total size. For each agent we have a maximum of 1000 time-step meaning that they can do 1000 different discrete moves per episode.

The sizes of the testing maps are $15x15$, $50x50$ and $100x50$ and the max time-step for the first case $15x15$ is 150 and 1000 for the other 2 cases. For the values of hyper parameters we kept steady most of them for the initial training phase and after that we tuned them for trying to optimize the agents behaviour. Also we had increased number of epochs based on the results of the agents. The table 4.1 shows the values of the hyper-parameters that we used. Finally, we slowly decreased the epsilon probability value and we also slowly increased the beta value as shown on the equations 4.1. We tested a lot of different values for the hyper-parameters and different optimizers (like Adam) but it had worse behaviour(results) so we ended up on the values of the 4.1.

We increased the number of epochs based on the final coverage of the maps. Specifically based on the table 4.1 if the coverage was above 40% we increased the epochs from 1 to 5(3) and if the coverage was above 60% we increased to 7(5) for the initial training phase(optimizing phase). This was done in order to accelerate the training and awarding the agents, when they had satisfying total map coverage.

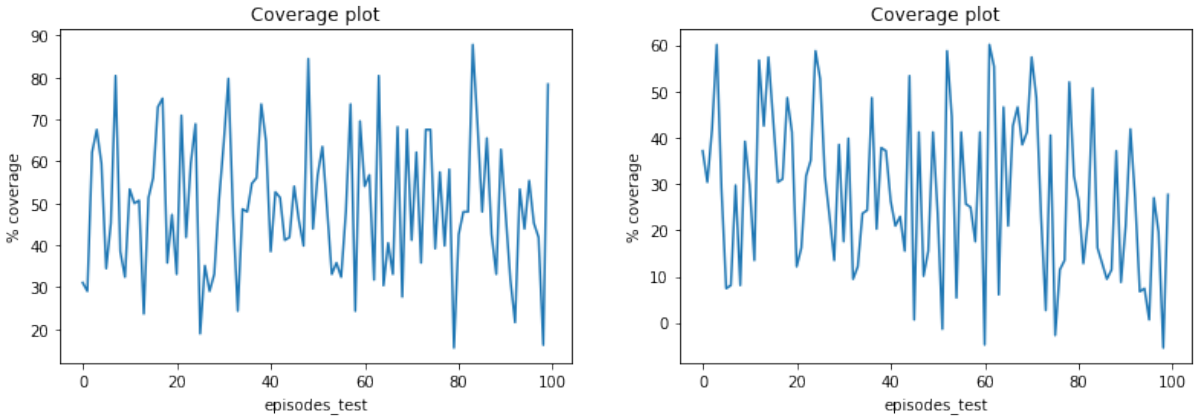
$$epsilon = min_{epsilon} + (max_{epsilon} - min_{epsilon})\frac{max_{exploration} - current_{step}}{max_{exploration}}$$
$$beta = min_{beta} + (max_{beta} - min_{beta})\frac{max_{exploration} - current_{step}}{max_{exploration}} \tag{4.1}$$

Equation 2: Decay of epsilon and beta

	Initial Training Phase	Optimizing Phase
Learning rate	0.00005	0.000005-0.000001
epochs	1,5,7	1,3,5
Batch size	32	64
Prioritizing Scale	0.55	0.7-0.8
Target Network Frequency	10000	10000
Memory buffer size	100000	100000
Optimizer	RmsProp	RmsProp
Loss Function	Huber Loss	Huber Loss
Epsilon Decay scale	Equation 4.1	0
Replay Steps	2	4
Beta increase	Equation 4.1	1

Table 4.1: Hyper parameter table

4.1 Different models on small maps



(a) DDQN model (b) dueling model

Figure 4.1: Two-agents individual paths

Firstly, we tested both neural structures that we presented in 3.3 in a 15x15 map. We used prioritized experience replay for our memory model. The figure **4.1** shows the coverage we had with 2 agents and the comparison with our main model. Specifically we see that with 2 agents we achieved a median of 27% in which is much lower that our classic model in which we have a median of 58%(not final trained model) for the 15x15

map. We tested also the dueling model for the 50x50 map but it had low coverage(12% for 1 agent) so we decided not to include any results for it. So, we concluded that the dueling model architecture is not proper for our problem and will be not used for the larger maps.

4.2 Multiple agent exploration on big maps

For the $50x50$ grid map case(random maps) and the $100x50$ (maze map) we show the individual pathings of the agents(different numbers) and the total coverage%. For the first case, we display the median coverage% and the number of wall hits(actions that resulted into a wall) for a high number of episodes in which we generated a random map each time. Also, we used the DDQN model for the following results.

4.2.1 Single and dual agent exploration

Both results can be seen in the table 4.3 and table 4.4. For our first test, we tested a single agent exploration on the $50x50$ map. The pathing is showed in the figure 4.2 in which the total coverage of the solo agent is 41%. The 4.11a shows the coverage of a single agent for the 100 testing episodes. The average coverage for those episodes is 35%.

Furthermore, we tested the dual agent cooperative exploration and we show the path of the agents for the 33%, 66% and 100% of total movement. The figures 4.3, 4.4 and 4.5 we see the path of the agents with the total map coverage being 62.9%. The table 4.2 corresponds to this case in which the first two columns shows the % of newly discovered of the respected total step and the next two shows the total discovery of the total step for each agent. The last one shows the total cover of the map that the agents make. Also, the results of the general testing for the two agents, for 100 episodes are being shown on the 4.11, with average coverage being 44%.

	Newly discovered agent-1	Newly discovered agent-2	Total discovered agent-1	Total discovered agent-2	Total coverage of the map
33%	84%	96%	84%	96%	27.32%
66%	64%	80%	74.46%	88%	47%
100%	60%	35%	69.44%	69%	63%

Table 4.2: 33%, 66% and 100% of total time step of the movement for 2 agents

4.2.2 Three and four agents exploration

Next, we tested on a different map the coverage for exactly 3 and 4 agents. The results for the both cases can be seen in the table 4.5 and table 4.6 in which the first column

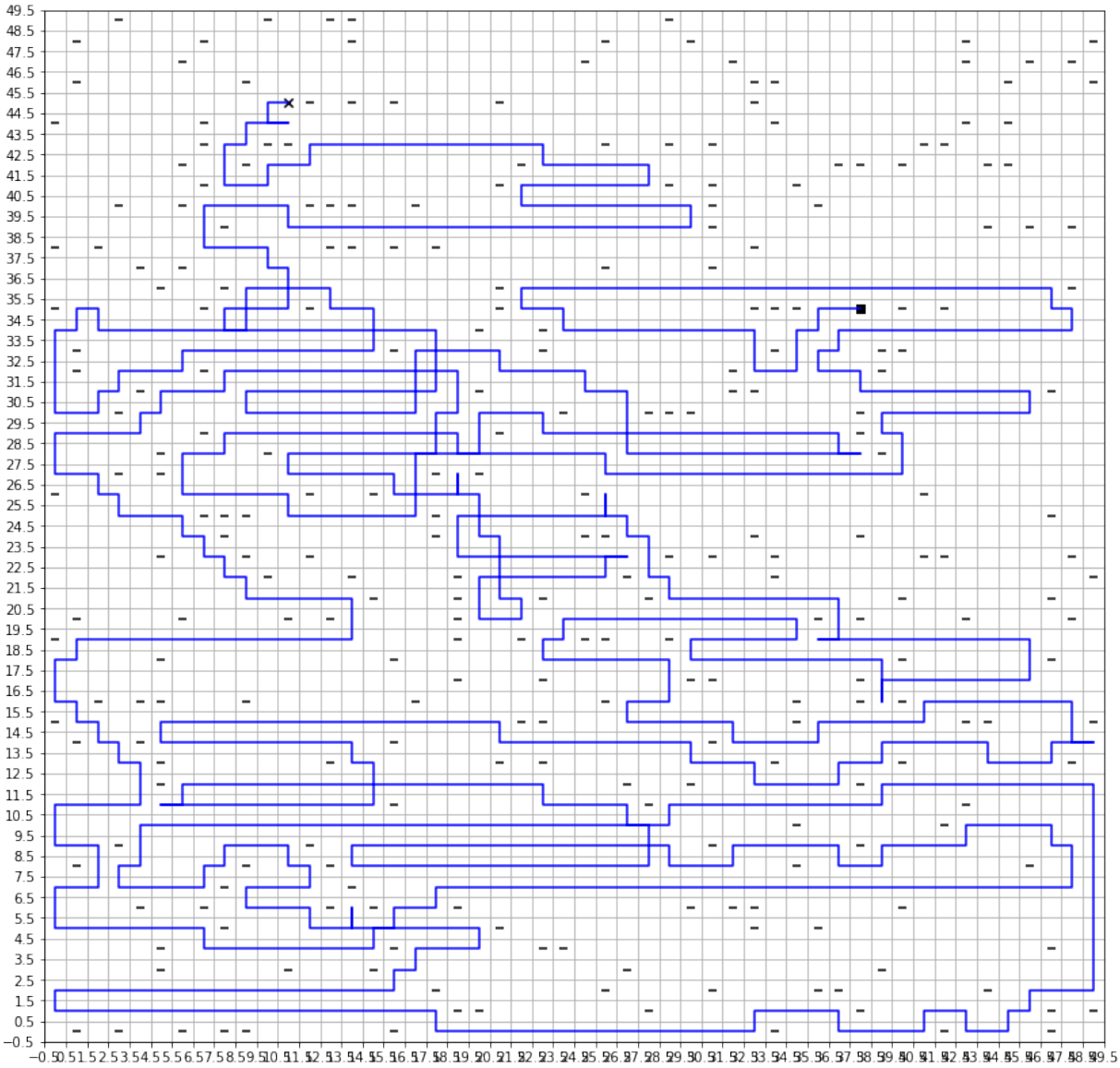


Figure 4.2: Single agent exploration

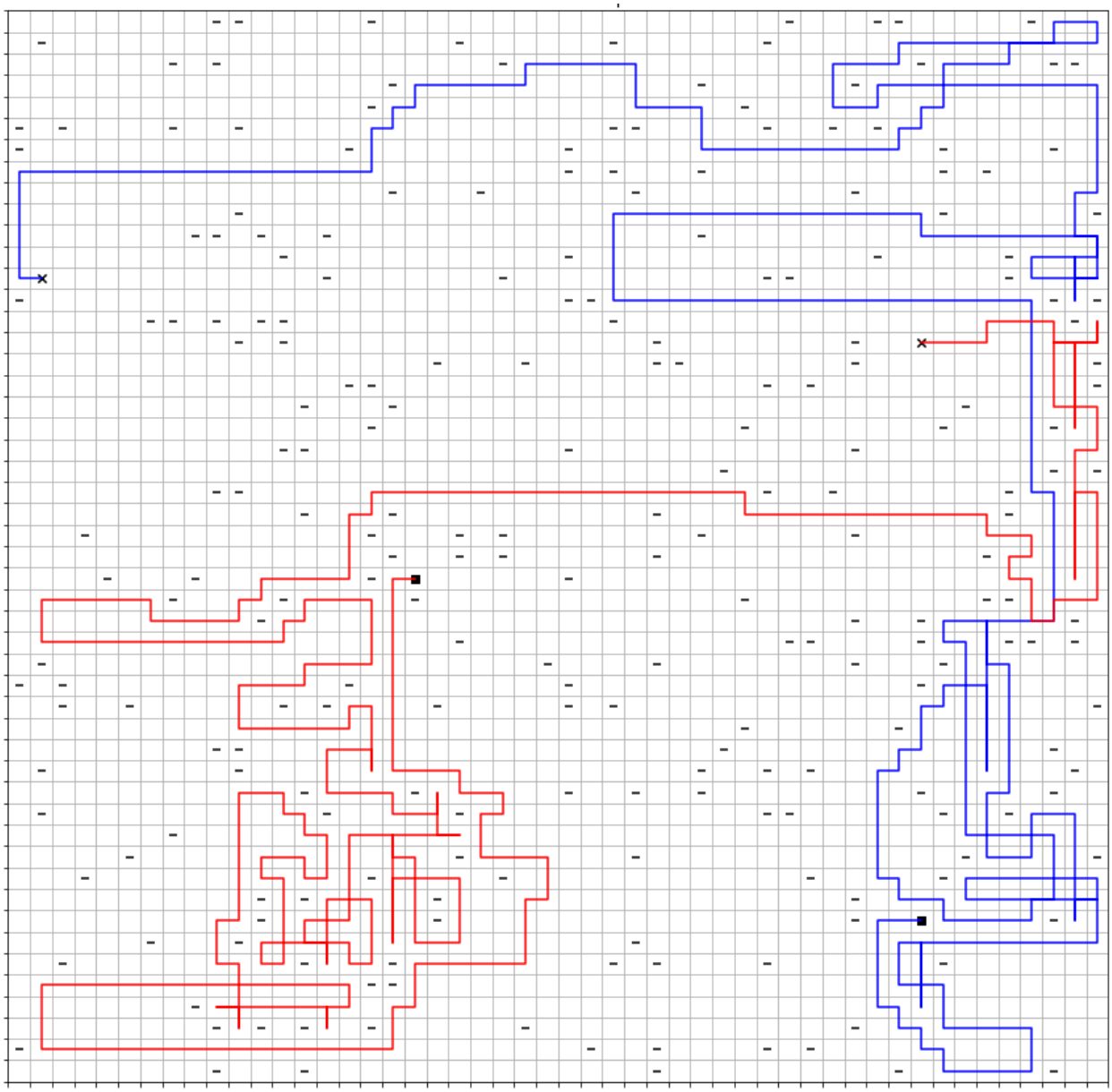


Figure 4.3: 33% of total time-step
2-agents exploration

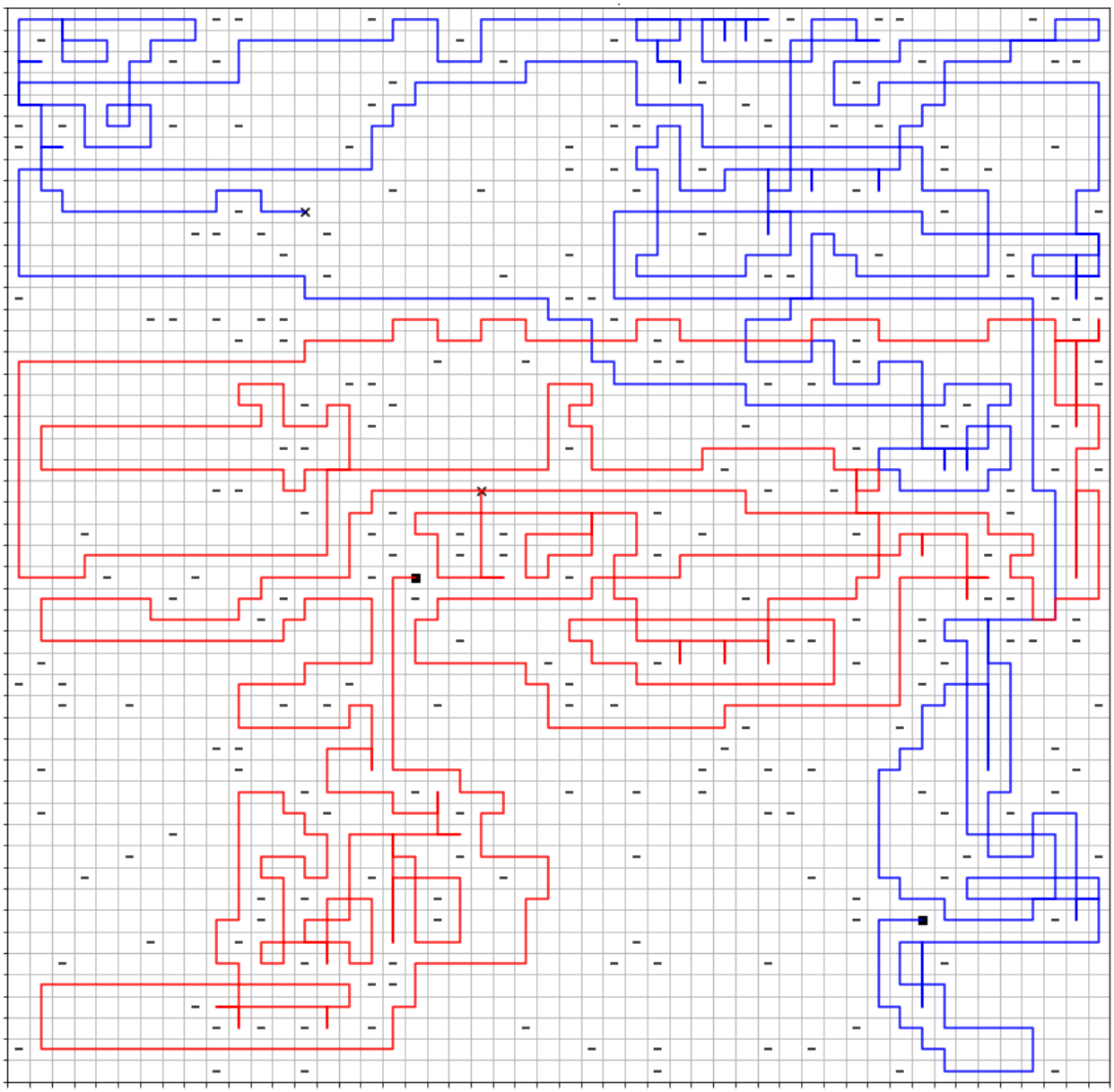


Figure 4.4: 66% of total timestep
2-agents exploration

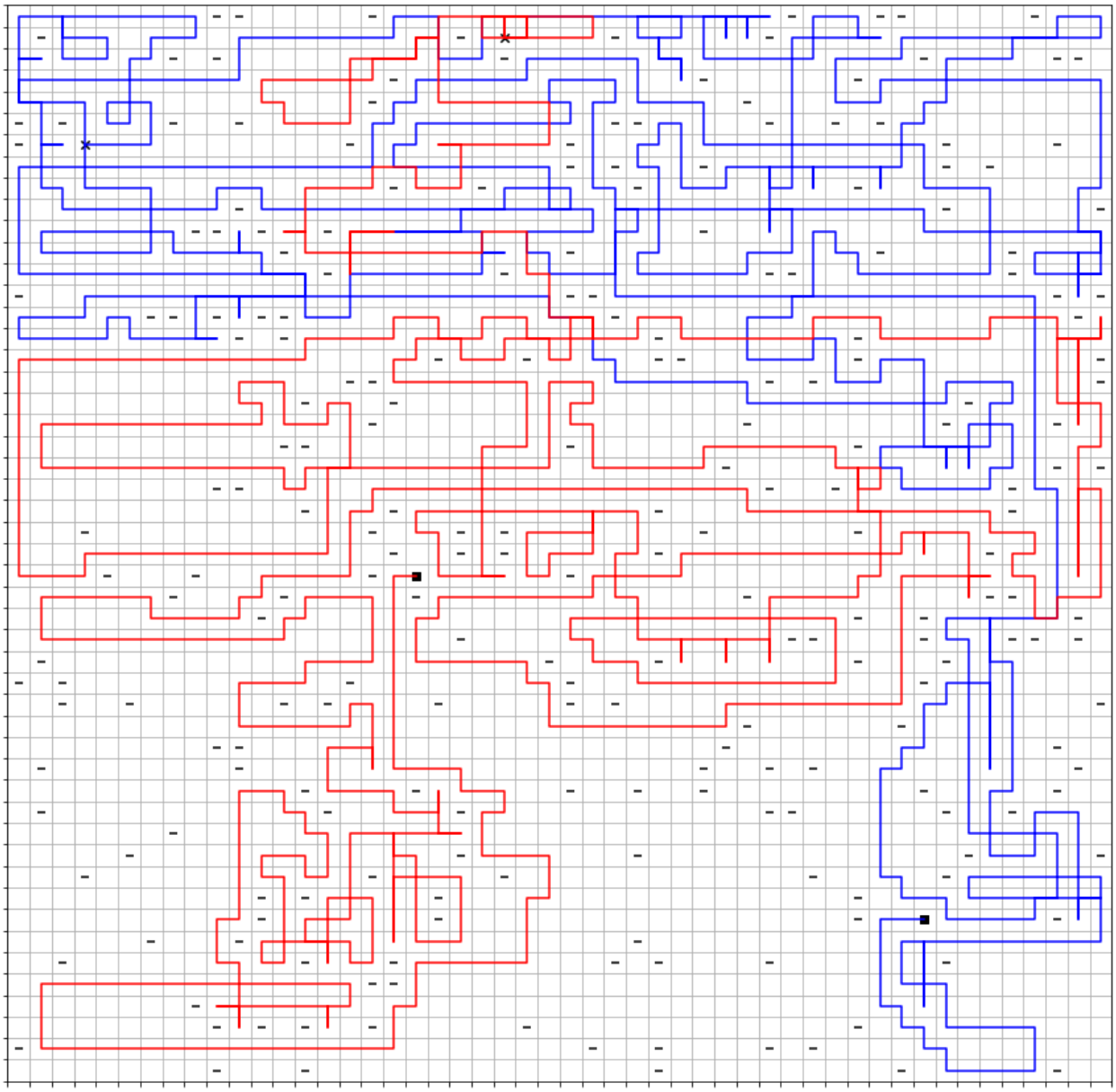


Figure 4.5: 100% of total timestep
2-agents exploration

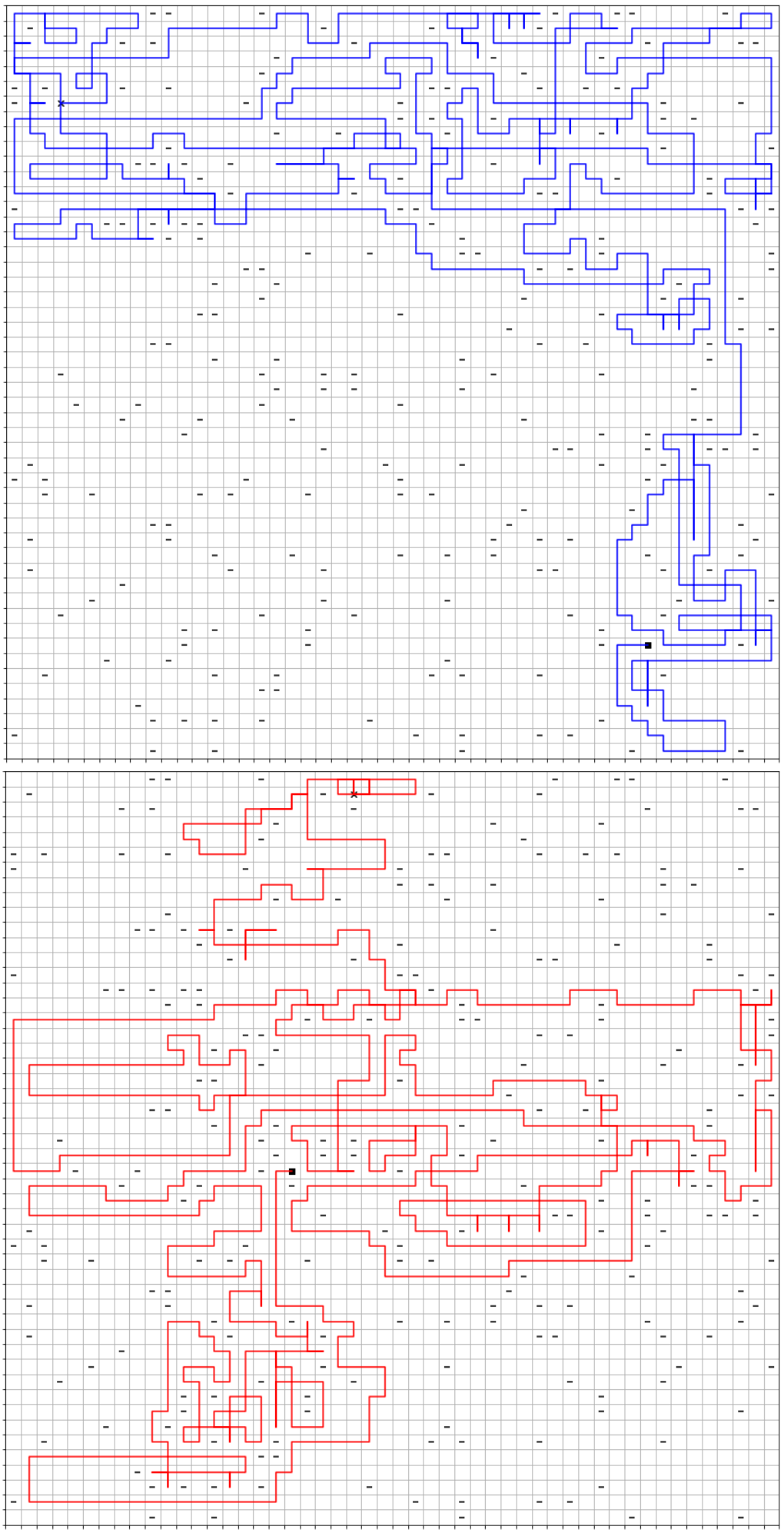


Figure 4.6: Two-agents individual pathing

shows their % coverage based on their entire movement(useful coverage) and the second their respected coverage% for the entire map.

For the three agent case the total coverage can be seen in the figure 4.7 and also the individual movements of the agents are being shown in the figures 4.8.

For the four agent case we show the total path coverage 4.9 and the each agent path in the figures 4.10. Finally, for the more general testing the 4.11 shows the coverage% for 100 episodes for different number of agents. The median coverage% for the three and four agents is 53% and 60% respectively. The number of actions that resulted into a wall is the same for all the different numbers of agents and it is 0.

	Useful coverage	Total coverage
agent-1 coverage%	90.2%	41%
Total coverage%	90.2%	41%

Table 4.3: Single agent coverage

	Useful coverage	Total coverage
agent-1 coverage%	69.44%	31.54%
agent-2 coverage%	69%	31.36%
Total coverage%	-	62.9%

Table 4.4: Dual agent coverage

	Useful coverage	Total coverage
agent-1 coverage%	46.5%	21.13%
agent-2 coverage%	43.7%	21.22%
agent-3 coverage%	49.9%	22.68%
Total coverage%	-	67.03%

Table 4.5: Three agent coverage

	Useful coverage	Total coverage
agent-1 coverage%	32%	14.54%
agent-2 coverage%	31.2%	14.18%
agent-3 coverage%	52.4%	23.82%
agent-4 coverage%	42.5%	20.43%
Total coverage%	-	72.97%

Table 4.6: Four Agents Coverage

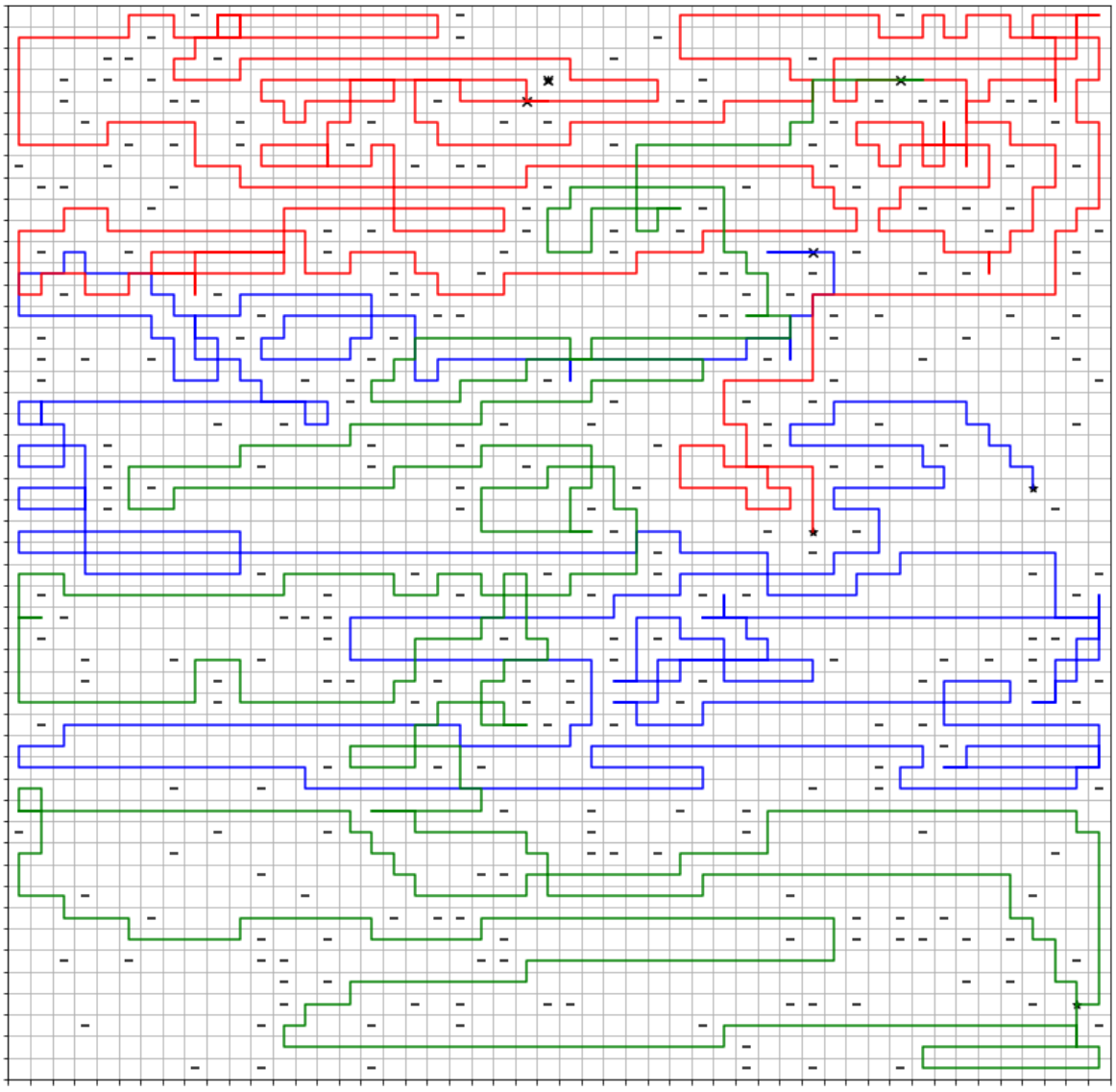
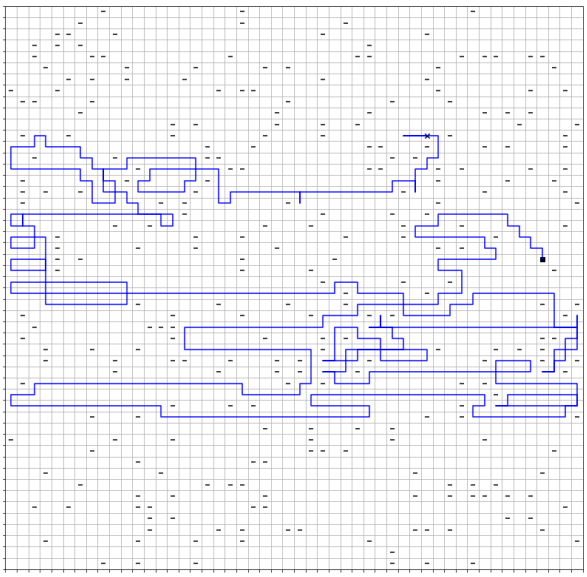
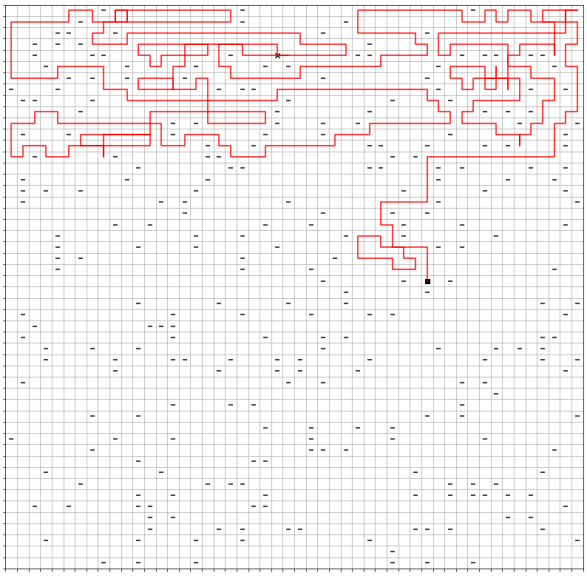


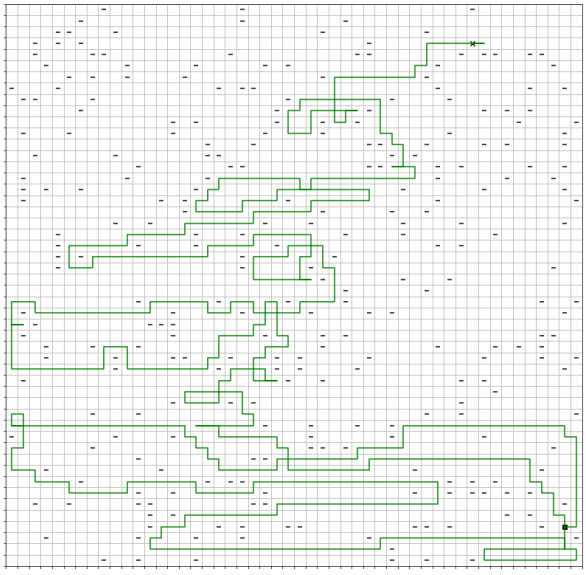
Figure 4.7: Three agents path



(a) agent-1 path



(b) agent-2 path



(c) agent-3 path

Figure 4.8: Three agents individual paths

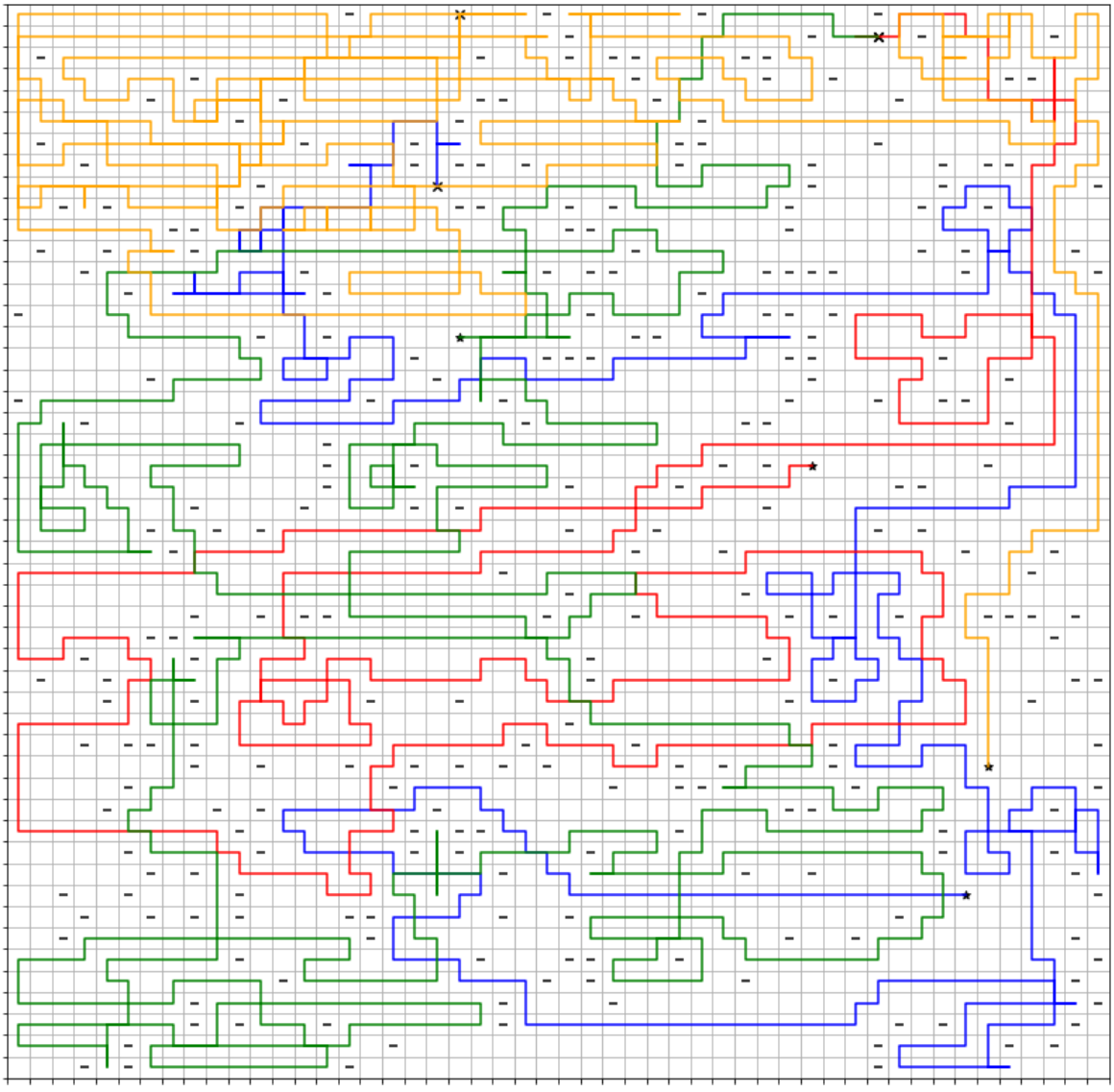
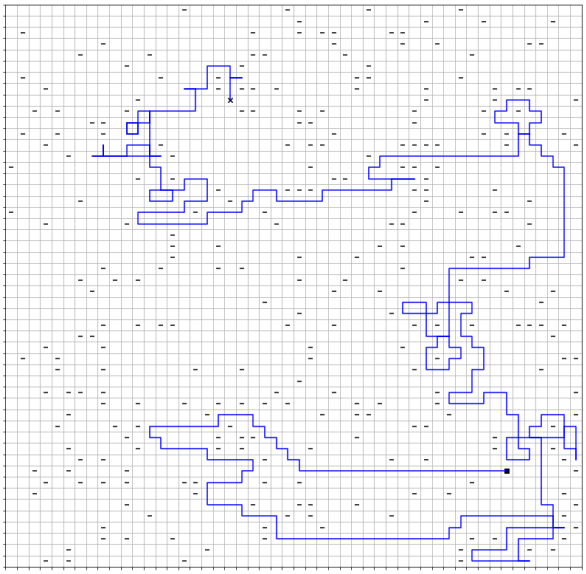
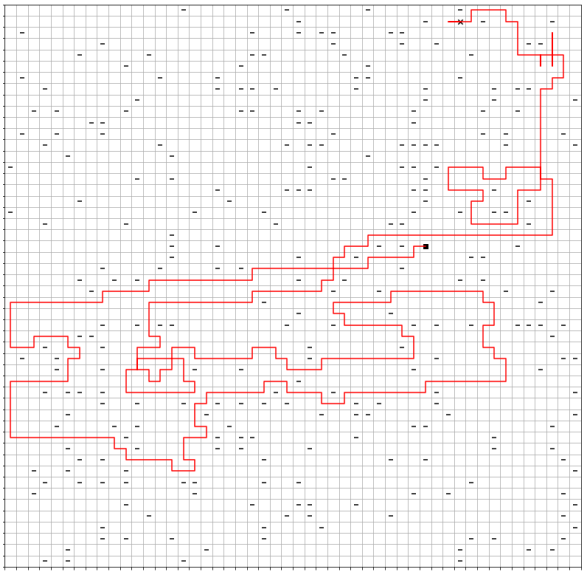


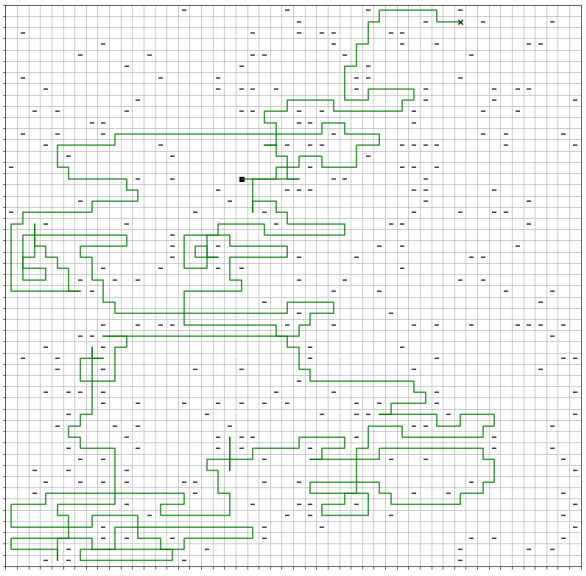
Figure 4.9: Four agents path



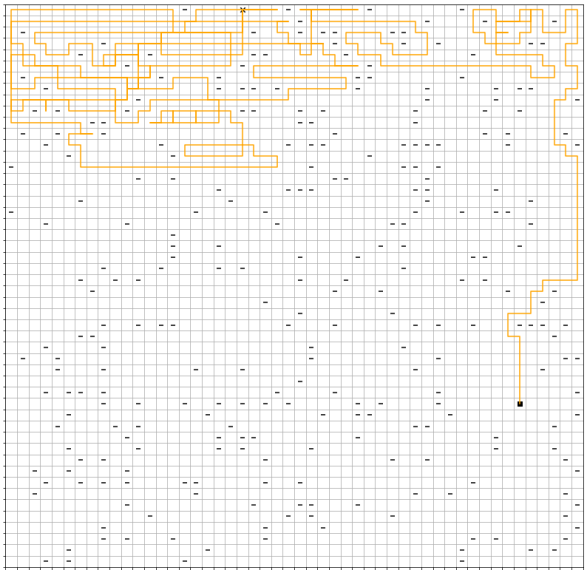
(a) agent-1 path



(b) agent-2 path

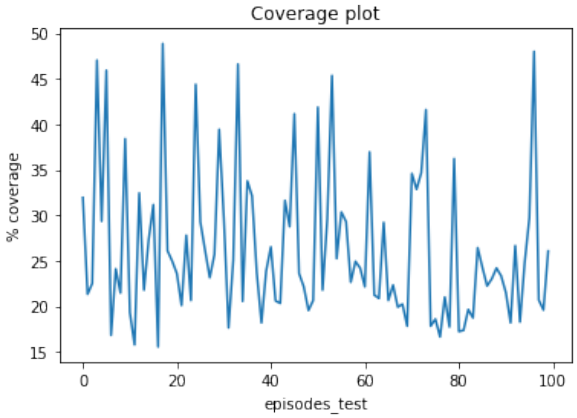


(c) agent-3 path

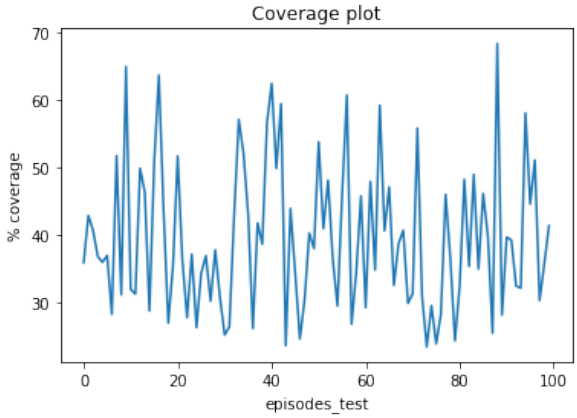


(d) agent-4 path

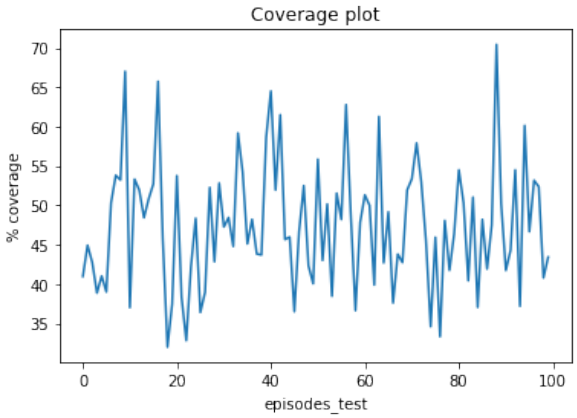
Figure 4.10: Four agents individual paths



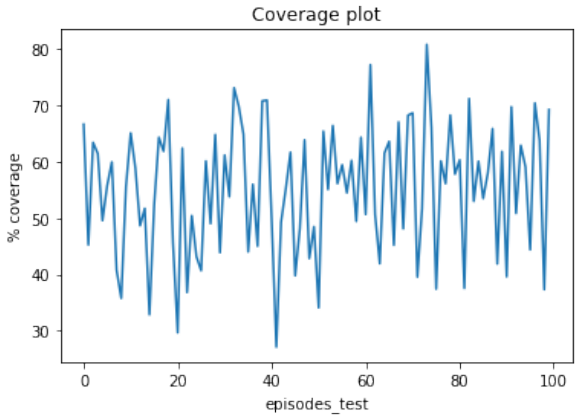
(a) Single agent testing



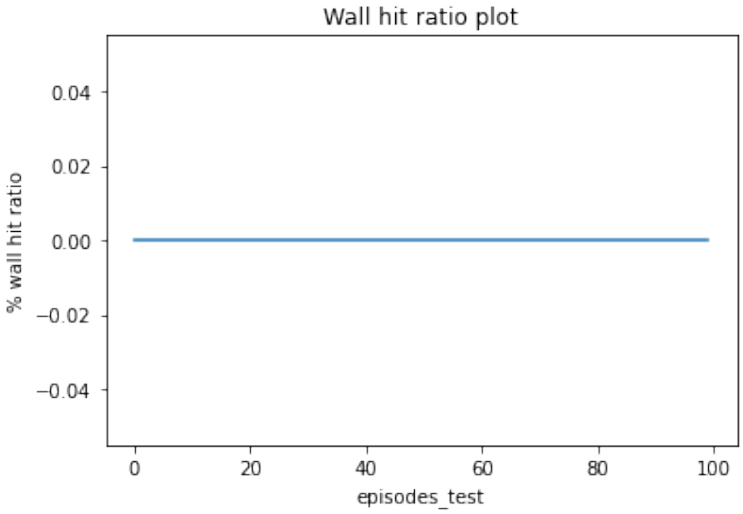
(b) Dual agents testing



(c) Three agents testing



(d) Four agents testing



(e) Wall hit ratio for all number of agents

Figure 4.11: Coverage and wall hit ratio

4.3 Maze maps exploration

Next, we evaluated our method for specific type of maze-map. Is is based on a real cave environment with a lot of dead ends and multiple small closed areas. We tested with two and four agents. With the two agents scenario we achieved a median coverage for 100 episodes testing(the only difference between each testing episode are the starting positions of the agents but the map is the same) a total(median) of 60% coverage. The simultaneous paths are being shown in the figure 4.12 and the individual paths are shown in the figures 4.13a and 4.13b.

Moreover, the path plan with 4 agents is being shown in the figure 4.14 and also the individual paths in the figures 4.15a ,4.15b, 4.15c, 4.15d. The median coverage for 100 episodes testing is 73%.

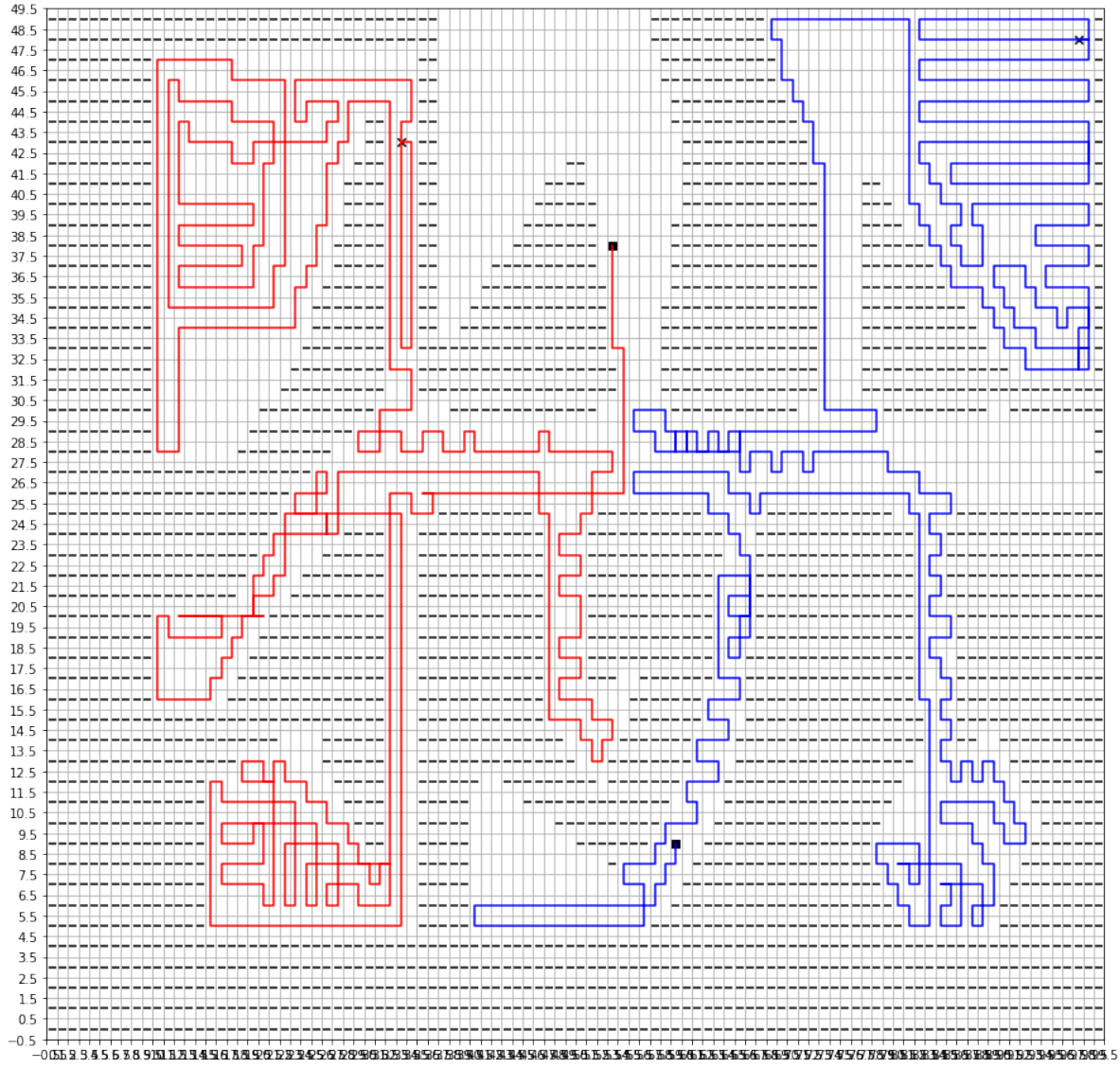
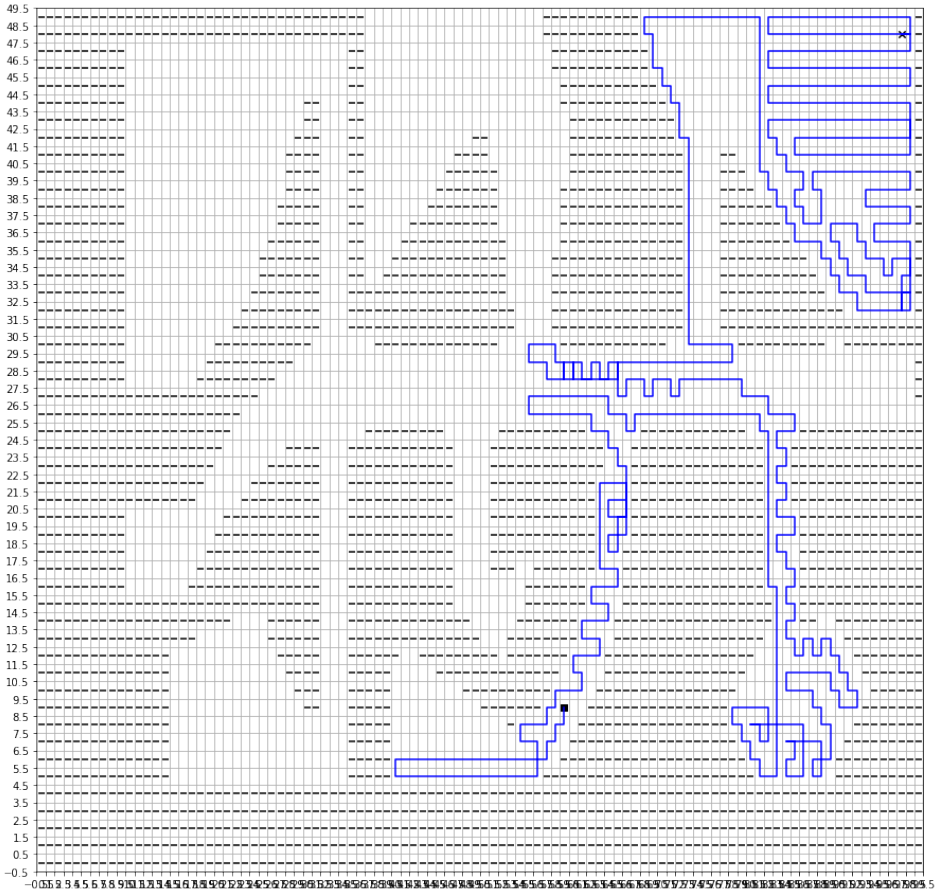
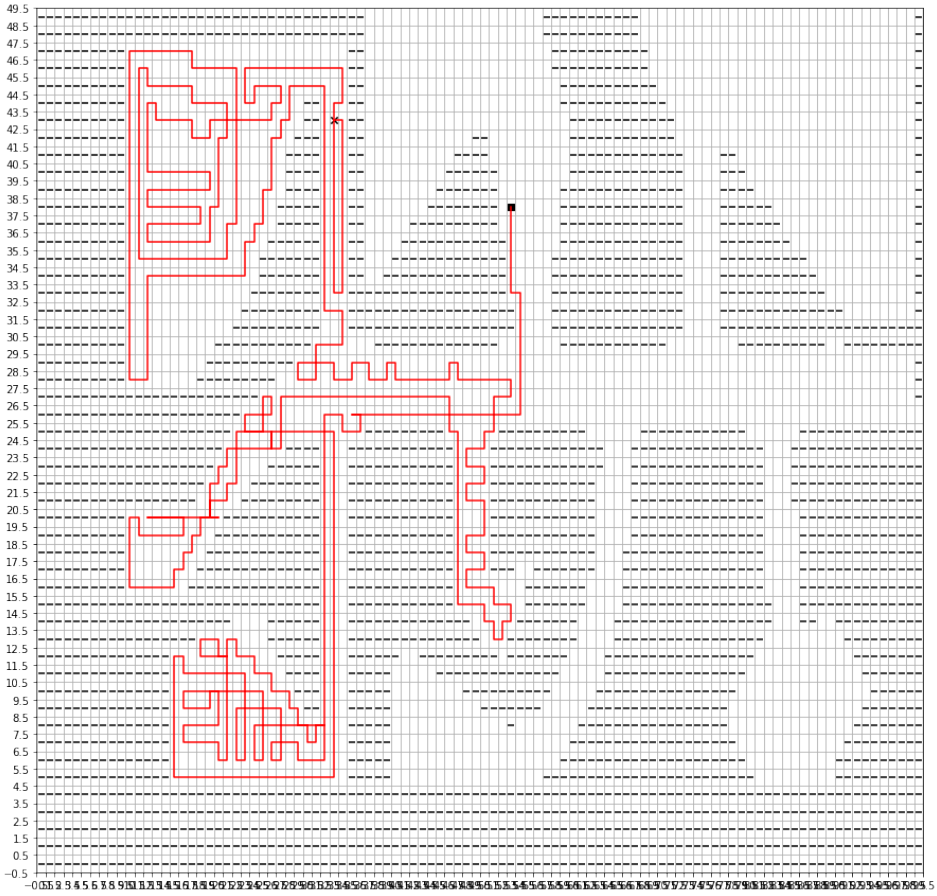


Figure 4.12: Maze map 2-agents
total coverage of 60%



(a) agent 1 path



(b) agent 2 path

Figure 4.13: Two-agents individual paths

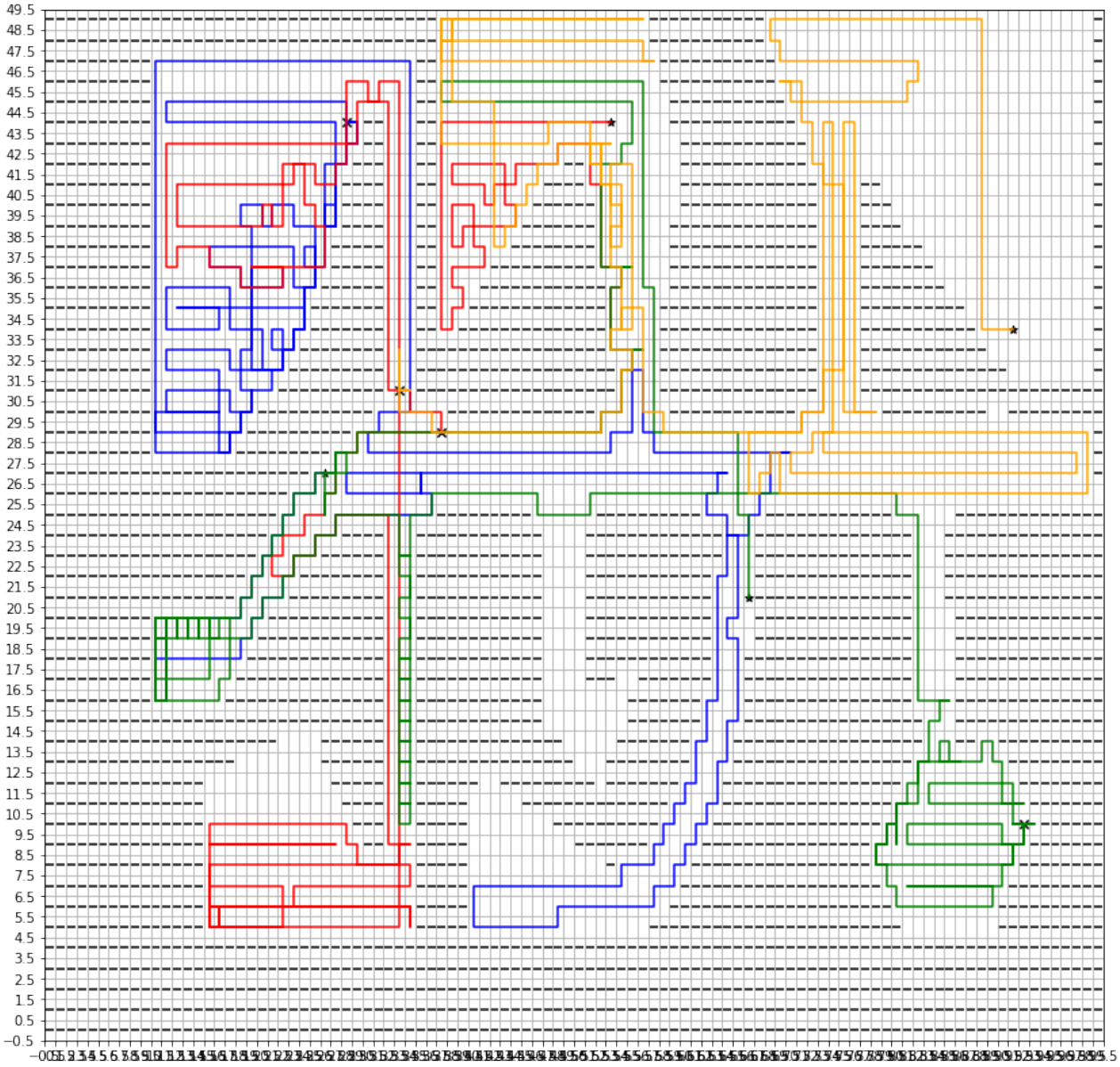
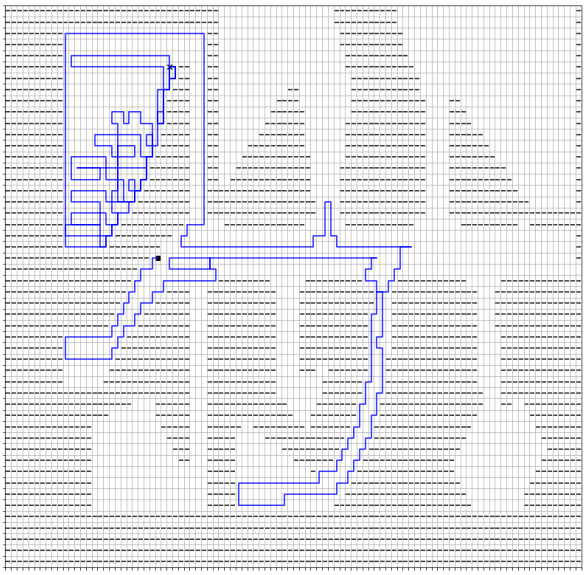
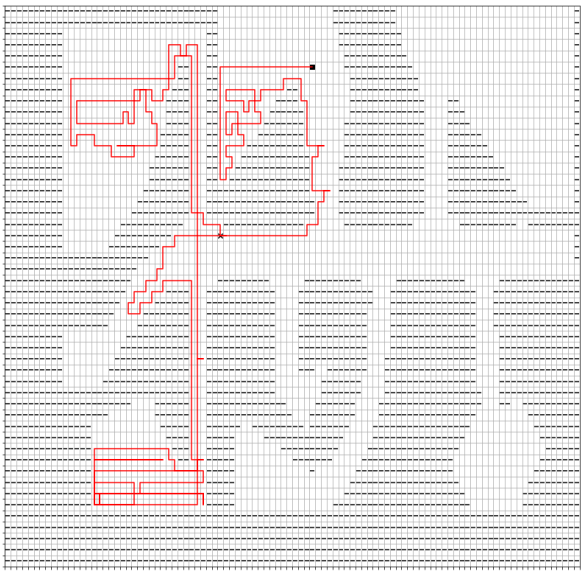


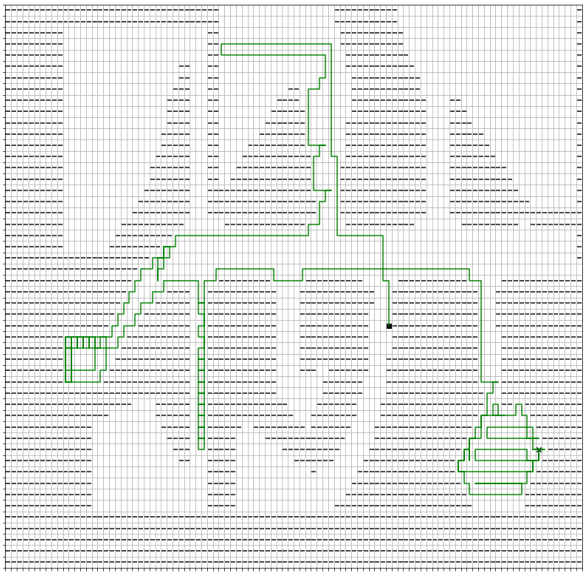
Figure 4.14: Maze map 4-agents
total coverage of 73%



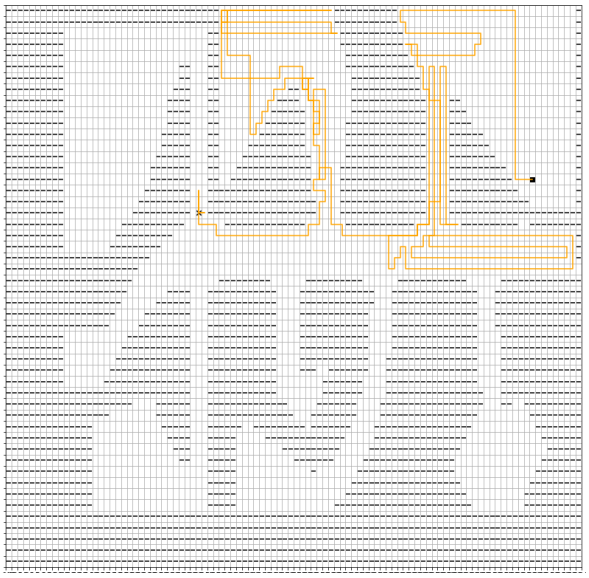
(a) agent-1 path



(b) agent-2 path



(c) agent-3 path



(d) agent-4 path

Figure 4.15: Four-agents individual paths for maze map

4.4 Different max time-step

Finally, we tested different maximum time-step for the agents. The agents are trained with a maximum of 1000 timesteps. Specifically we show the coverage we have with 500 and 2000 value of timestep. We see that an agent with a specific pre-trained max-timestep struggles to achieve good behaviour on larger timestep (we were expecting close to the double coverage for the double max time-step).

timestep=500	useful coverage	Total coverage
agent-1 coverage%	98%	22.27%
Total coverage%	-	22.27%

Table 4.7: Single agent coverage for 500 time-step

timestep=1000	Useful coverage	Total coverage
agent-1 coverage%	90.2%	41%
Total coverage%	-	41%

Table 4.8: Single agent coverage for 1000 time-step

timestep=2000	Useful coverage	Total coverage
agent-1 coverage%	55.22%	48.5%
Total coverage%	-	48.5%

Table 4.9: Single agent coverage for 2000 time-step

Chapter 5

Discussions

5.1 Importance of time-step

For all the training (and optimizing) phase each episode ended , when the max time-step was occurred. We didn't have any terminal case like end the episode when agents achieved 100% coverage in order to simulate real life exploration scenario on unknown maps. The coverage of the entire map is based of how many squares of the grid map the agent have discovered. So raising the max-time step of the agents from 1000 moves to a higher value we would expect that the agent(s) would have higher coverage. Specifically doubling the time step from 1000 to 2000 the coverage raised between (5-10)% for the single agent scenario. We had similar results with more agents. This suggests that the agents behaviour towards exploration is strictly connected with the time-step but further research is needed to understand the importance of it. So, for our approach it makes sense to have pre-trained agents with different max time-steps for different size of maps for a near life scenario. There is no need to know the exact size of the map but an approximation is fine ,as our testing suggested that. Also, the half value max time-step(500) worked as expected, since it had almost half the coverage of the 1000 timestep.

5.2 Total coverage for different number of agents and time-step.

In the table 5.1 we see the different coverage we had in our testings(random created maps 50x50 size). We see that the exploration with more than 1 agent isn't raising linearly. The biggest gain we have is from 1 to 2 agents with a 21.9% raise in exploration. From two to three agents we had 6.56% raise and from three to four 8.96% raise. Future research must be done in order to optimize the number of agents for the different map sizes.

Finally, we have very good coverage on the maze map scenario and also we see co-operative behaviour of the agents meaning they try to explore different areas. This may

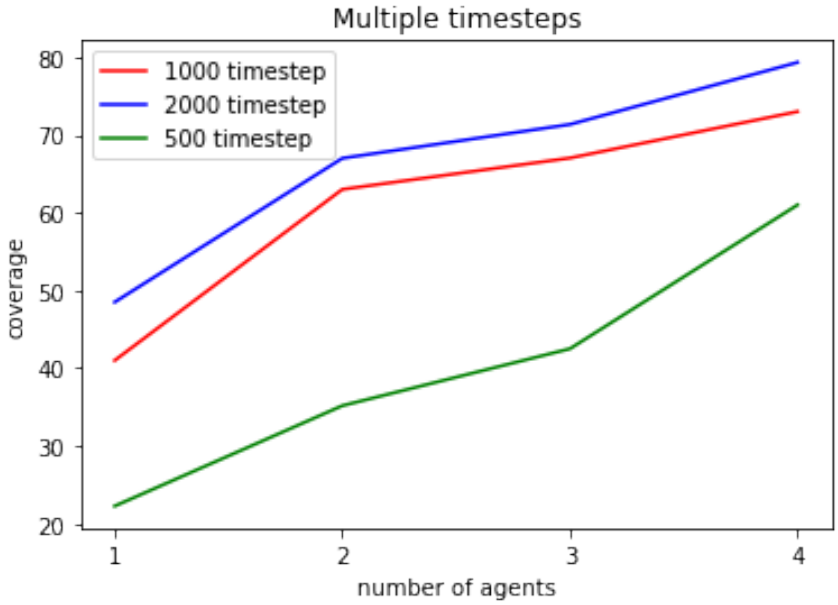


Figure 5.1: Multiple step agents

happens due to the sharing information of the gps coordinates but further research is needed in order to understand the cooperative behaviour from the reinforcement learning approach.

	Total coverage
Single agent exploration	41%
Two agent exploration	62.9
Three agent exploration	67.03%
Four agent exploration	72.97%

Table 5.1: Median coverage of all agents on the 50x50 (random)maps

Chapter 6

Conclusions

In this study, we introduced a reinforcement learning approach for multi-agent systems in an unknown environment. Although its a 2D approach, we can easily up-scale it to 3d for real life applications, like in inspection, search and rescue missions, and the generally for exploration of unknown maps. Furthermore, we showed the different coverage with different number of agents but further investigation is required in order to find the optimal number of agents for the different size of maps. We successfully eliminated the need of previous knowledge of the maps, which corresponds to the real life problems. Generally, we observed cooperative exploration, meaning that the agents tried to explore different areas, that aren't necessarily close between them in order to optimize exploration. Also we achieved zero chance of collision of an agent into a wall. Lastly, we observed a little bit of oscillation regarding the total coverage of the agents. This is happening due to the nature of the used algorithm (RL) and also from the randomness of the created environments and the starting positions of the agents. Maybe a much grater size of the neural net could reduce the amplitude of the oscillation but this needs more research. However, the RL algorithm has shown satisfactory results for the decision making of the agents and the general solution for the exploration of unknown environment.

Bibliography

[1] A. Wasik, J. N. Pereira, R. Ventura, P. U. Lima, and A. Martinoli, "Graph-based distributed control for adaptive multi-robot patrolling through local formation transformation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ieee, 2016, pp. 1721–1728.

[2] G. Cubber, D. Doroftei, K. Rudin, K. Berns, D. Serrano, J. Sanchez, S. Govindaraj, J. Bedkowski, and R. Roda, "Search and rescue robotics-from theory to practice," in *Search Rescue Robotics From Theory to Practice*. IntechOpen, 2017.

[3] S. Montambault and N. Pouliot, "Design and validation of a mobile robot for power line inspection and maintenance," in *Field and Service Robotics*. Springer, 2008, pp. 495–504.

[4] J. Das, G. Cross, C. Qu, A. Makineni, P. Tokekar, Y. Mulgaonkar, and V. Kumar, "Devices, systems, and methods for automated monitoring enabling precision agriculture," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2015, pp. 462–469.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[6] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, pp. 2613–2621, 2010.

[7] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[8] M. Wu, Y. Gao, A. Jung, Q. Zhang, and S. Du, "The actor-dueling-critic method for reinforcement learning," *Sensors*, vol. 19, no. 7, p. 1547, 2019.

[9] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1, pp. 113–126, 2001.

[10] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset, "Limited communication, multi-robot team based coverage," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 4. IEEE, 2004, pp. 3462–3468.

[11] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1, pp. 77–98, 2001.

[12] M. A. Batalin and G. S. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *Distributed Autonomous Robotic Systems 5*. Springer, 2002, pp. 373–382.

[13] W. Chang, W. Lizhen, Y. Chao, W. Zhichao, L. Han, and Y. Chao, "Coactive design of explainable agent-based task planning and deep reinforcement learning for human-uavs teamwork," *Chinese Journal of Aeronautics*, 2020.

[14] Z. Liu, B. Chen, H. Zhou, G. Koushik, M. Hebert, and D. Zhao, "Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," *arXiv preprint arXiv:2007.15724*, 2020.

[15] K. Wan, X. Gao, Z. Hu, and G. Wu, "Robust motion control for uav in dynamic uncertain environments using deep reinforcement learning," *Remote sensing*, vol. 12, no. 4, p. 640, 2020.

[16] Y. Yang, K. Zhang, D. Liu, and H. Song, "Autonomous uav navigation in dynamic environments with double deep q-networks," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–7.

[17] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[18] W. B. Knox and P. Stone, "Combining manual feedback with subsequent mdp reward signals for reinforcement learning," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. Citeseer, 2010, pp. 5–12.

[19] M. Coggan, "Exploration and exploitation in reinforcement learning," *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.

[20] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[21] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[22] M. B. Hafez and C. K. Loo, "Topological q-learning with internally guided exploration for mobile robot navigation," *Neural Computing and Applications*, vol. 26, no. 8, pp. 1939–1954, 2015.

[23] J. Peng and R. J. Williams, "Incremental multi-step q-learning," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 226–232.

[24] S. B. Thrun, "Efficient exploration in reinforcement learning," 1992.

[25] I. Rekleitis, A. P. New, and H. Choset, "Distributed coverage of unknown/unstructured environments by mobile sensor networks," in *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 2005, pp. 145–155.

[26] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein, "Distributed covering by ant-robots using evaporating traces," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 918–933, 1999.

[27] J. Hu, H. Zhang, L. Song, Z. Han, and H. V. Poor, "Reinforcement learning for a cellular internet of uavs: Protocol design, trajectory control, and resource management," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 116–123, 2020.

[28] K. S. Aitken, M. D. McNeil, S. Hermann, P. C. Bundock, A. Kilian, K. Heller-Uszynska, R. J. Henry, and J. Li, "A comprehensive genetic map of sugarcane that provides enhanced map coverage and integrates high-throughput diversity array technology (dart) markers," *BMC genomics*, vol. 15, no. 1, pp. 1–12, 2014.

[29] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang, and G. Hollinger, "Multi-uav exploration with limited communication and battery," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 2230–2235.