Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Project in Intelligent Systems - DVA439

APPLICATION OF REINFORCMENT LEARNING TO THE MAZE SOLVING PROBLEM

Dženita Škulj
dsj19001@student.mdh.se

Lamija Hasanagić
lhc19001@student.mdh.se

Tin Vidović
tvc19001@student.mdh.se

Lejla Murselović
lmc19002@student.mdh.se

Professor: Ning Xiong
ning.xiong@mdh.se
Mälardalen University, Västerås, Sweden

September 26, 2020

Abstract

This paper looks into the application of a Reinforcement Learning algorithm on a classical maze solving problem. A Q-learning algorithm, as a special type of Reinforcement Learning algorithm, is implemented in the Python programming language in order to model and solve the maze. A detailed description of the Q-learning algorithm and its adaptation to the maze solving problem is given. The analysis of the obtained results is then presented focusing on the speed of the convergence of the algorithm to the optimal solution for different sizes of the maze and different parameters of the algorithm. Finally, a conclusion is provided with insights for potential interesting future work.

Table of Contents

1. Introduction

The maze solving problem is an ancient well-known problem solved by humans and animals alike. The problem is defined as finding the shortest path from the starting point of a maze to its end point. A problem like this has multiple paths that go from the starting point to the end point, but the required solution is usually the one with the minimum path length. Since mazes can be easily represented as a tree data structure in a mathematical sense, we call mazes graph theory problems. The solutions of graph theory problems in general is very useful both from a theoretical standpoint as well as a practical one. Maze solving problems in particular have always been interesting problems to solve in an optimal way, as solving them enables interesting use in real world applications, for example for navigating a robot through its environment. This paper looks into the application of reinforcement learning techniques on solving such maze problems, specifically in the setting of a maze solving game.

Reinforcement learning is one of three basic machine learning techniques, alongside supervised learning and unsupervised learning. In supervised learning the agent is trained against a known correct solution, whereas in reinforcement learning and unsupervised learning there is no known optimal solution. In unsupervised learning the emphases lies on analyzing clusters of input data, while in reinforcement learning an agent is positively or negatively reinforced (rewarded or punished) for its actions. Through this system of rewards and punishments the agent learns the optimal solution through iterations of trial-and-error runs called episodes. Learning is largely dependent on exploring the environment and subsequently exploiting the information gathered in the exploration phase. Reinforcement learning, particularly, has been used to solve a variety of problems like multi-robot cooperation, AI game play in PC games, robotic manipulator control, soccer robot teams, training the models that control autonomous cars etc. [1].

There are four main classes of reinforcement learning algorithms in particular: the Sarsa, temporal difference, function approximation and the Q-Learning algorithms. In this paper a Q-Learning algorithm is chosen to model and solve a maze. Q-Learning was proposed by Watkins et al. in [2]. The Q-Learning algorithm and its application to the maze solving problem considered throughout this paper is described in detail in Section 4. [1].

In this project the Q-Learning algorithm is implemented for moving an agent (rat) through a maze, from its initial starting position, to the final position in the maze, which is represented by a piece of cheese. This algorithm is implemented as a Python application that visualizes the maze and the movement of the rat, throughout several episodes, where the rat's behaviour improves from episode to episode. This is due to the *learning property* of the implemented algorithm. This means that the rat will improve its behaviour by gradually exploring the maze and learning from this exploration according to the principles of Q-Learning, until it finds the shortest path. Once the rat has found the shortest path through the maze, i.e. the Q-Table converges to the optimal solution, it can exploit the Q-Table it generated to play the maze game in an optimal way.

2. Related Work

2.1. Lejla Murselović

In the paper by Gunady and Gomaa [3] a Maze Solving Problem is is approached with Reinforcement learning, but on top of that they use a generalization technique on the Reinforcement learning algorithm. Reinforcement learning (RL) algorithms depend on generating lookup tables that will store the state-action pairs. In this project the chosen RL algorithm is Q-learning algorithm and it is storing the state-action pairs in a Q-table. Having to use lookup tables increases the computation and space complexity of RL, especially for real life problems that have a large state-action space. Q-learning poorly rely on experience, so the convergence rate is low. Time and storage requirements become to demanding. Introducing a more compact representation of the state-action space is crucial for many practical applications. If a generalization technique is applied the the learning process becomes feasible. Using *state aggregation* in there generalization,

60% of state space was reduced and a 4x higher speed of convergence rate was achieved.

The paper by Osmanković and Konjicija [4] is a proof-of-concept showing how the Reinforcement learning is used on the Maze Solving Problem. This paper introduces a MATLAB implementation. For the performance analysis *time of execution* and *total number of iterations* were measured. The simulation involved several different dimension mazes (5x5, 7x7, 9x9, 11x11, 13x13,15x15, 17x17, 19x19 and 21x21). For each maze the simulation is run 10 times and the average was taken for both tests; time of execution and number of iterations. This paper showed that the number of iterations linearly increases with the maze size $O(n)$, while the time of execution increases, rather quadratically or cubically with the maze size, $O(n^2)$ or $O(n^3)$.

2.2. Dženita Škulj

In [4] . The paper shows an abstract implementation of Q-learning that aims the maze solving. The paper considers the interaction between the agent and environment using a matrix representation of the Q table. This work is done in order to confirm the theoretical complexity of the algorithm. Authors are using time and number of iterations to present an analysis of an algorithm that is written in the Matlab. The results are done for different sizes of maze such as 5x5, 7x7, 9x9, 11x11, 13x13, 15x15, 17x17, 19x19, and 21x21. Also, the results are investigated for average measurements of the considered sizes of the maze. As a conclusion, the fact is that the number of iterations linearly increases with the size of the maze, while the time is a non-linear function that could be quadratic or cubic.

Advanced usage of Q-learning algorithm is proposed in paper [5]. The paper proposed the method combined of the Q-learning and artificial neural networks for finding the shortest path in the maze. In the paper, the authors are describing the impact of adding the artificial neural network on the Q-learning algorithm and the efficiency of the Q-learning algorithm. For assessing the efficiency of the algorithm, the testing is done on the same maze for the proposed method and simple q-learning. Tests are done on the maze of size 15x15. They considered the number of steps in the function of trials. Hence, the conclusion is that combining the artificial neural network with a simple Q-learning algorithm gives 4-5 times better in the time sense results in finding the shortest path.

The paper [6] gives a detailed overview of Reinforcement Learning, Q-learning and Approximate Q-learning algorithm. The paper provides the results of the experiment in order to test the performance of the Q-learning versus Approximate Q-learning algorithm. The experiment presents the performance of algorithms trough episodes. Analyzed results show that the performance of Approximate Q-learning is higher than the simple Q-learning algorithm.

2.3. Lamija Hasanagić

This paper focuses on the application of the reinforcement learning algorithm, which is known as Q-learning, to the maze solving problem. Research in the area of reinforcement learning, as it is stated in [1], started in the late 80s and early 90s and has made a breakthrough progress since then. In this paper exhaustive survey of different reinforcement learning algorithms, including Q-learning, as well as the model of reinforcement learning are provided, which can be used as a good basis for work to be done in the context of this paper. Further overview of the Q-learning algorithm specifically is presented in [7]. The author covers crucial principles in the Q-learning algorithm such as value functions, optimal policies and exploration vs. utilization rate and as such can be seen as a useful theoretical background to Q-learning.

When comes to the application of Q-learning to the maze solving problem, the paper written by Osmankovic and Konjicija [4] can be seen as a good starting point. In it, the authors map a maze matrix to a Q-table and implement the Q-learning algorithm in Matlab to find the path form the top left corner to the bottom right corner. The work done in this paper is similar in this sense, but aims to analyze the effect of different parameters and different exploration rate decay in addition

to the size of the mazes on the convergence rate.

More advanced topics are covered in several papers. For example, Yu et al. [8] in their work use a specialized camera at the bottom of the vehicle that is able to take a picture of the maze. The maze solving problem obtained in this way is then solved using Q-learning algorithm. This allows UGV (Unmanned Ground Vehicle) to move through the real space. They also propose improved epsilon-greedy strategy in comparison to the one given in our paper. Hacibeyoglu and Arslan [5] in their paper present the application of Q-learning algorithm that is improved with artificial neural network on search and maze problems. The results obtained in this paper show that this new approach allows an agent to learn 4 to 5 times quickly in the completely unknown environment in comparison with the approach when only Q-learning algorithm is used.

The scalability issues that is highlighted in this work as well, is addressed in [3]. In this paper the authors proposed one way of dealing with complexity and size of the state-action space. They proposed a new algorithm called SAQL which first searches through all the states and then attempts to aggregate similar states into super states, thereby reducing the size of state-action space considerably. The results show that SAQL converges to the solution faster than the original Q-learning algorithm especially for the mazes of larger sizes.

2.4. Tin Vidović

A good overview of the most common Reinforcement Learning algorithms is given by Qiang and Zhongli in [1]. The paper summarizes the main principles and compares the Sarsa, Temporal Difference, Function Approximation and Q-Learning algorithms and in doing so provides a conceptual overview of the strengths, weaknesses and potential applications of each one.

The main principles behind the Q-Learning algorithm are explained in [7]. The main principles such as modeling the environment through a Markov Decision Process and using Q-Learning to derive the optimal policy are introduced.

The paper that could be seen as a basis to the work conducted in this paper is [4]. In it Osmankovic and Konjicija, implement the Q-Learning algorithm in MATLAB in order to solve a maze problem similar to the one being solved in our paper. The analysis of the results however is only done from the perspective of the size of the maze being solved. The work done in this paper aims to expand on this, by looking at different learning rate values and different decaying of the exploration rate.

Several improvements of the fundamental Q-Learning algorithm can be made, specifically for the case of solving mazes and graph problems in general. For example, the Q-Learning algorithm, in the way it is implemented in this paper, uses a basic $\epsilon - greedy$ strategy in which the agent explores the environment more in the early stages and less in the latter stages. This exploration rate decays slowly through iterations either exponentially or linearly. Yu, Wu and Sun [8] propose an improved strategy and show it greatly reduces the randomnesses component in exploration vs. exploitation. In [9] Q-Learning is compared to Fuzzy Rule Interpolation based Q-Learning for maze solving problems and it is shown that FRIQ-learning provides better convergence rates compared to the original Q-Learning algorithm. Another area of research seeks to expand Q-Learning algorithms with artificial neural networks to solve more complex and dynamic problems.

A big issue in applying Q-Learning to problems such as the maze solving problem is the issue of scalability. Specifically Q-Learning algorithms do not scale well with larger state-action spaces. In [3] the authors propose a new algorithm, dubbed SAQL, which aggregates similar states into one state. The results obtained show that using SAQL instead of the original Q-Learning algorithm provides faster convergence rates for maze solving problems. Finally, in [10] a method for reshaping rewards obtained by the agent in maze solving problems is proposed. Based on extra information such as which states are bad, certain sub-states and sub-environments it is shown that a large gain in the average cumulative reward per iteration that is obtained by the agent is possible.

3. Problem Formulation

As stated in the Introduction this paper looks into the application of Reinforcement Learning, specifically the Q-Learning algorithm on the *classic* maze solving problem.

Here we define the *classic* maze problem as a problem of finding a path from the top left corner to the bottom right corner of a square maze. One *move* can be made at each time step, by moving in one of the four possible directions: up, down, left or right. A move is only legal if the tile being moved to is free, i.e. not outside of the maze and not a wall. The maze is considered to be solved upon reaching the bottom right corner. The score is awarded according to the number of steps needed to solve the maze, where less steps equals a larger score.

With this definition of a classic maze problem in mind the research done in this paper focuses on answering the following Research Question:

- **RQ1:** *How can Reinforcement Learning be used to solve the classic maze solving problem?*
- **RQ2:** *How is the efficiency of the algorithm affected by different learning rates, sizes of the maze and exploration rate decay algorithms?*

4. Approach and method

As mentioned previously, this paper provides an implementation of a Q-learning algorithm as a special case of Reinforcement Learning algorithms to the *classical* maze solving problem. The *classic* maze problem is a problem of finding a path from the top left corner to the bottom right corner of a square maze. One *move* can be made at each time step, by moving in one of the four possible directions: up, down, left or right. A move is only legal if the tile being moved to is free, i.e. not outside of the maze and not a wall. The maze is considered to be solved upon reaching the bottom right corner. The score is awarded according to the number of steps needed to solve the maze, where less steps equals a larger score. The implementation of the algorithm is done using the Python programming language. The program visualizes the maze environment as well as the movement of the rat through the maze. The maze is represented by a square matrix of different proportions. The light grey tile represents the mouse, starting in the top left corner, while the darker grey tile represents the goal or the cheese, located in the bottom right corner. The white tiles represent free tiles, to which the rat can move to, while the black tiles represent walls, which are inaccessible to the rat. An example of a maze is given in Figure 1.

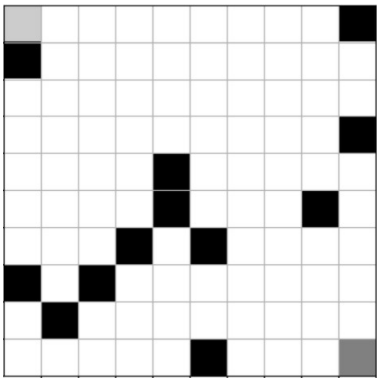

Figure 1: Maze
way the

In order to understand the Q-Learning algorithm, the main characteristics of the algorithm and their application to the maze solving problem are discussed in the following subsections.

4.1. Markov Decision Processes

Markov Decision Processes (MDPs) provide us with a simple way to formalize sequential decision making. This formalization enables structuring problems that are then solved with reinforcement learning. The main components of an MDP are: an *agent* (rat) that interacts with the *environment* (maze) which it is placed in. These interactions of the agent with the environment occur sequentially over many time steps. At each time step the agent uses its sensors to observe the *state* of the environment. In the maze solving problem the state of the environment can be seen as the location of the rat within the maze in a certain time step. Based on this perceived state the agent selects an *action* (up, down, left or right) and performs it. By performing the action the environment moves to a new state and the agent receives a *reward* or a *punishment*.

Throughout this process the agent attempts the maximize the total amount of rewards it receives. We say that the agent is attempting to maximize its *cumulative rewards* that it receives over time.

In other words an MDP has a set of states **S**, a set of actions **A** and a set of rewards **R**. in our case each of these sets has a finite number of elements. At each time step **t**, the agent learns it is in a state S_t. Based on this information it selects an action A_t. This is called the state-action pair (S_t, A_t). We can think of the process of receiving the reward or the punishment as a mapping of the state-action pairs to rewards. At each time step **t** we have:

$$f(S_t, A_t) = R_{t+1}$$

4.2. Expected Return

As stated previously, the goal of an agent in a Markov Decision Process is to maximize its cumulative rewards. In order to explain the concept of the cumulative rewards, the concept of *expected return* must be introduced. The expected return, in a simple way, can be seen as a sum of the all future rewards. Let us define the return G at time t as follows:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T$$

where T is the final time step. So, the agent's goal is to actually maximize the expected return of rewards. In our case the expected return is the sum of all rewards that the agent expects to receive from the current state until it reaches the end of the maze.

4.3. Policies and Utility Functions

Taking into consideration all the possible actions that an agent is allowed to take in all possible states of its environment the question of probability that an agent will select a specific action from a specific state becomes very interesting. The concept of *policies* is the one that addresses this question and represents one of the main concepts of Reinforcement Learning alongside Markov Decision Process and expected return. A policy can be defined as a function that maps a given state to probabilities of selecting each possible action from that state. If we denote a policy that agent follows as $\pi(s)$, then at a certain point in time, denoted as t, $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. In other words, the probability that the agent will take action a from the current state s at time t is $\pi(a|s)$, if agent follows the *pi* policy.

On the other hand, utility functions, which are as well as policies one of the main concepts of Reinforcement Learning, address the question of how good any given action or any given state is for the agent. Otherwise, utility functions are functions of states or state-action pairs that estimate how good it is for the agent to perform an action in a given state. There are two types of utility functions that must be taken into consideration: state-utility functions and action-utility functions.

The state-utility function, for a certain policy π, addresses the question of how good is any given state for an agent that follows the policy π. In other words, the value of state s under policy u_π can be defined as the expected return under policy π starting from state s at time t:

$$u_\pi = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

The action-utility function, for a certain policy π, addresses the question of how good is for the agent to take a specific action among many possible actions from a current state under policy π. In other words, the value of action a in state s under policy π can be defined as an expected return that starts from state s, at time t, takes the action a, and follows the policy π after that:

$$q_\pi = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

4.4. Optimal Policies

The optimal solution (policy) to the given problem is a solution that yields the the highest expected utility and is denoted by π^*. The expected utility generated by policy π from state is given with the equation:

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] \tag{1}$$

where $U(s)$ is the expected utility, $U^\pi(s)$ is the utility function for policy π, γ is the discount factor that describes the preference for recent rewards over future rewards as a number between 0 and 1, $R(s)$ is the reward function and $R(S_t)$ is is the reward value for the current state S_t in time step t. The optimal policy is mathematically given with the following formula:

$$\pi_s^* = \underset{\pi}{argmax} U^\pi(s) \tag{2}$$

$$\pi_s^* = \underset{a \in A(s)}{argmax} \sum_{s'} P(s'|s,a)U(s') \tag{3}$$

The recursive *Bellman equation* that states that the utility of a current state is the reward of that state summed with the expected discounted utility of the following state is used to find the optimal policy. The Bellman equation solutions are the same unique solutions of 1. The recursive Bellman equation is given with:

$$U(s) = R(s) + \gamma \cdot \underset{a \in A(s)}{max} \sum_{s'} P(s'|s,a)U(s') \tag{4}$$

4.5. Q-learning Algorithm

The *Q-learning* algorithm employs a *Q-learning function* that gives the expected utility for each action in each state. Q-learning is a type of *active learning* because the agent must interact with the environment as much as possible in order to learn how to optimally behave in it. Instead of learning the utility (state-utility) function $U(s)$, the agent learns the best action-quality (action-utility) function $Q(s,a)$. The relation between the utility and the learning function is simple:

$$U(s) = \underset{a}{max} Q(s,a) \tag{5}$$

Finding the learning function $Q(s,a)$ is an iterative process that relies on the Bellman equation:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \cdot \underset{a'}{max} Q(s',a') \tag{6}$$

A Q-Table is constructed in this project to hold these pairs. In our implementation the rows of the table represented all the possible states e.g. the 9 tiles in a 3x3 maze, therefore the number of rows will be n^2, if n is the size of the maze. The columns represented each of the actions up, down, left or right. An example of a Q-Table generated for a 3x3 maze is given in Figure 2.

Whenever a transition from state s to s' happens doing action a, the new iteration updates the appropriate Q-Table value as follows:

$$Q(s,a) = Q(s,a) + \lambda(R(s) + \gamma \cdot \underset{a'}{max} Q(s',a') - Q(s,a)) \tag{7}$$

where λ is the learning rate parameter and γ the discount rate. Once the learning function $Q(s,a)$ is found, it is easy to find the policy, mathematically given as:

$$\pi_s = \underset{a}{argmax} Q(s,a) \tag{8}$$

The agent's goal is to get to the cheese in the shortest amount of time, that is to find the shortest path to the cheese. This would, in a mathematical sense mean to find the optimal policy π_s^*. The agent finds the optimal policy throughout a number of episodes. An episode is defined by its start that is the initial state of the maze and its end that is when the agent has reached a maximum number of actions taken or found the cheese.

At the beginning the maze is unknown to the agent and this is represented with all zero values in the Q-table and the agent has to learn about its environment through *exploration*. In the latter stages the agent should rely more on the information it gathered so far in order to *exploit* its environment. In the project an ϵ greedy strategy is employed. This means that an exploration rate parameter is initialized to 1. In each time step a uniformly random number R is generated in the range [0, 1]. If this number is greater than the exploration rate, the rat will choose a random action independent of the Q-Table in order to explore its environment. If the number is less than the exploration rate, the rat simply chooses the action that gets him the highest reward based on the Q-Table. After each episode the exploration rate decays. Two decay rates were used and compared in this project: linear decay rate and exponential decay rate. The ϵ greedy algorithm ensures that the rat will explore its environment more in the early stages when the exploration rate is closer to 1, and exploit the environment as the exploration rate approaches 0.

Finally, in our project, using two rewards with the values -100 and $+5000$ proved most efficient. The reward, or punishment in this case, for taking a new action is set to -100. This punishment is introduced to motivate the agent to find the shortest path and not wander around aimlessly. The other reward in the set of rewards is $+5000$, and the agents receives this reward when it finds the cheese, that is when it takes the action that brings him to the bottom right corner of the maze.

When all episodes have finished, the optimal policy should have been found. in order to play thee agent simply executes actions recommended by the optimal policy, found in the Q-Table, in each time step. We say the agent *follows* the policy.

Q-TABLE	UP	DOWN	LEFT	RIGHT
1	0	-10	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	-10
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0

Figure 2: Q-table

5. Results and analysis

The performance of the implemented algorithm is analysed from several different perspectives. As a first step, the way at which the number of steps per each of the episode changes, was analyzed.

In Figure 3 blue markers represent the number of steps that the rat took in each of 3000 episodes in a 20x20 maze. The curve fitting through the number of steps for each of the episodes was used to show how the algorithm converges throughout the episodes as the rat learns to solve the maze.

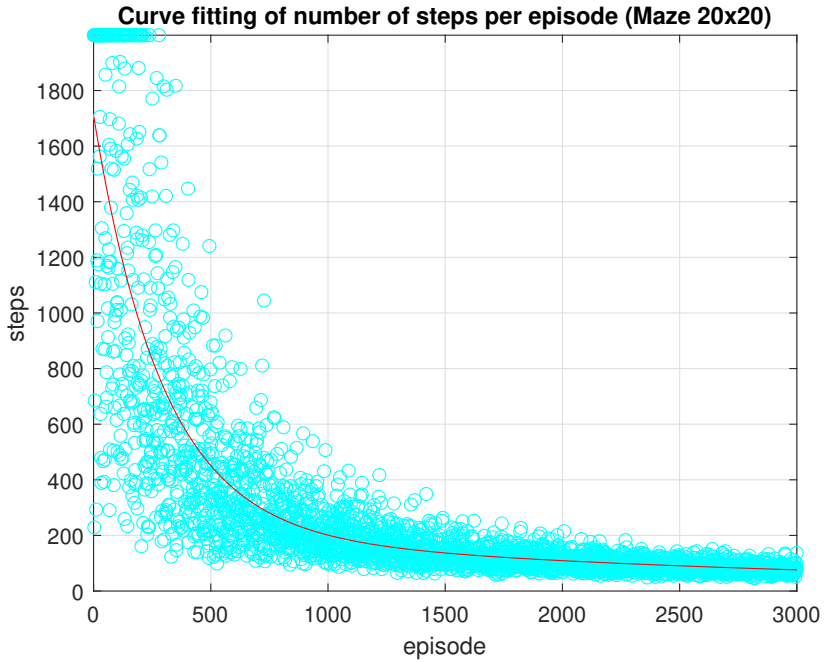

Figure 3: Curve fitting through number of steps per each of the episodes - Maze 20x20

Throughout the analysis it was noticed that the size of the mazes is obviously a large factor when it comes to the efficiency of the algorithm. The efficiency of the algorithm is measured with mazes of five different sizes: 5x5, 8x8, 10x10, 15x15 and 20x20. The number of steps is averaged for each 25 episodes and the following plots are produced. All other parameters such as the learning rate and exploration rate decay are left unchanged. As can be seen in Figure 4 for the smaller mazes, the convergence to the solution is reached in less episodes than is the case for the larger size mazes.

Another performance metric measured is the speed of convergence for different learning rates. In (7) the learning rate is introduced and it controls how much the agent values new information over old one. In other words if the learning rate is equal to 1, the old q-table values are overwritten with new q-values in each iteration, whereas if the learning rate is equal to 0, only old q-table values are retained. Thus the learning rate is usually a number in between 0 and 1. Figure 5 shows the number of steps per episode for a larger, 25x25 maze and four different learning rates 0.3, 0.6, 0.7, 0.9. As can be seen from the figure, the learning rate does not play a central role in the speed of convergence to the solution of the Q-learning algorithm.

The final measurement presented concerns the algorithm for the decay of the exploration rate ϵ. Two standard decay rates are measured: linear decay and exponential decay. The linear decay is given with: $\epsilon = 1 - \alpha n$ and the exponential decay is given with: $0.01 + (1 - 0.01) * e^{\alpha * n}$, where alpha is a constant between 0 and 1, and n is the current episode number. As can be seen from Figures 6, 7 and 8 the exponential rate proved better for smaller size mazes, while the linear decay rate performed better for larger mazes.

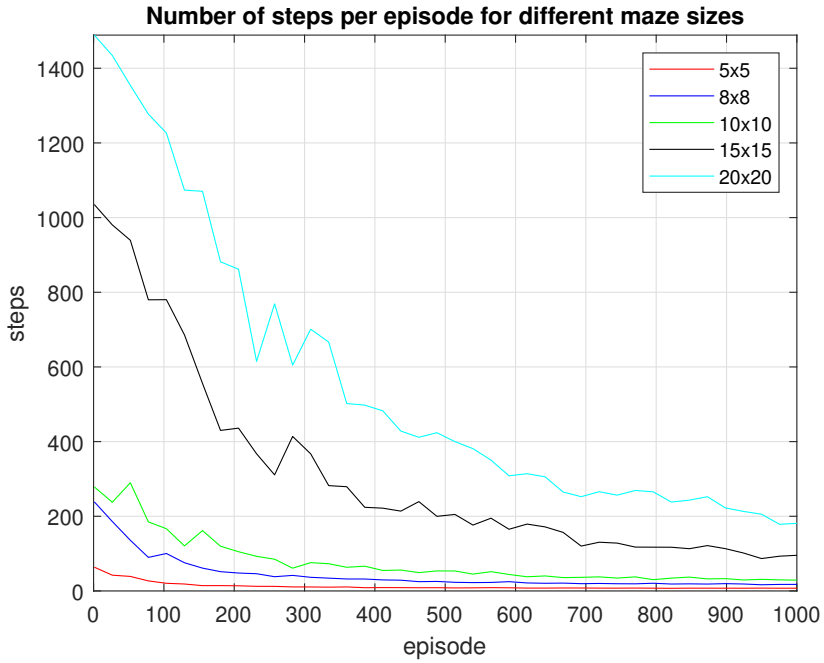

Figure 4: Number of steps per each of the episodes for different maze sizes

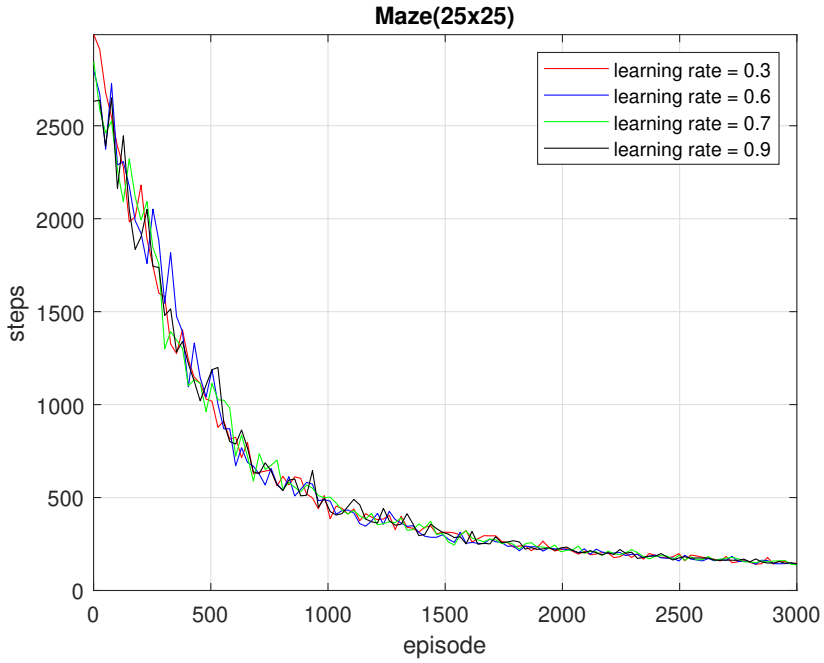

Figure 5: Number of steps per each of the episodes for different learning rates - Maze 25x25

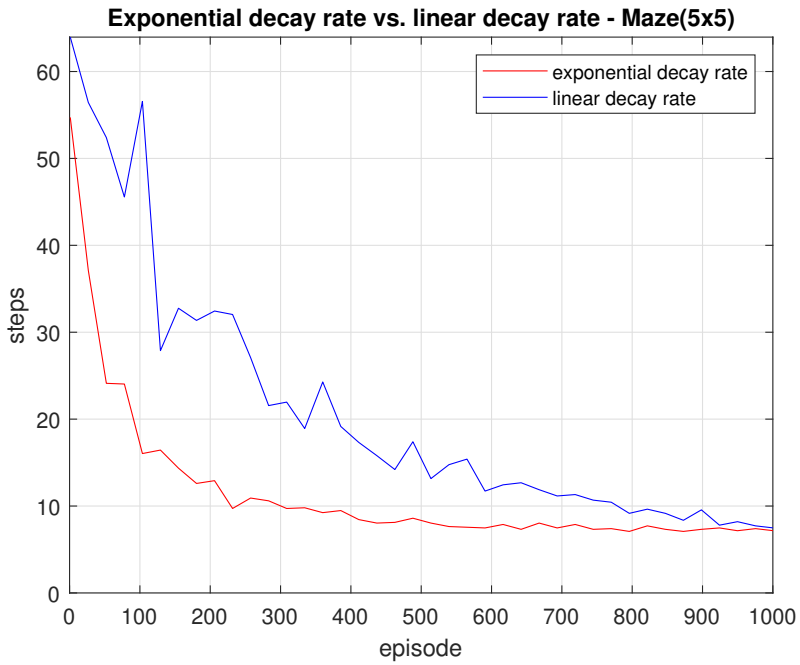

Figure 6: Exponential decay rate vs. linear decay rate - Maze 5x5

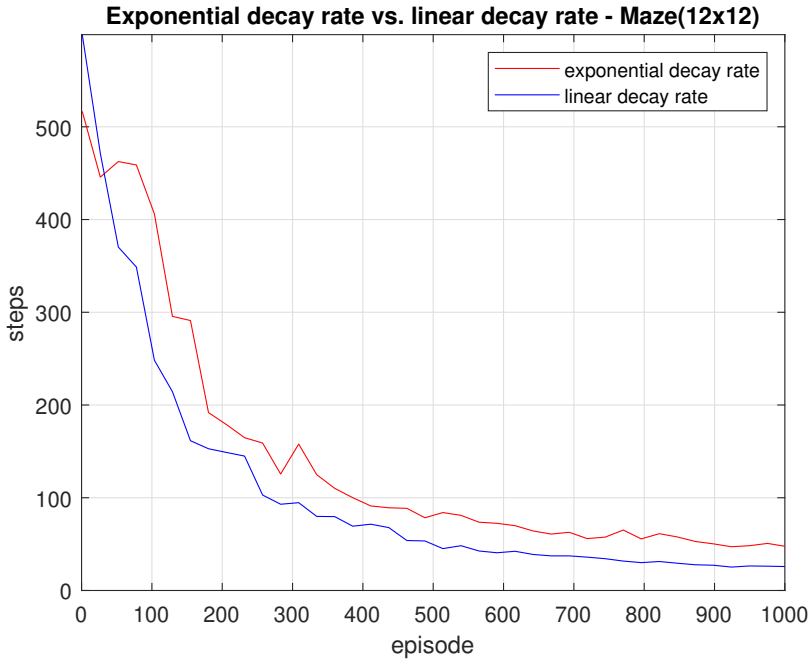

Figure 7: Exponential decay rate vs. linear decay rate - Maze 12x12

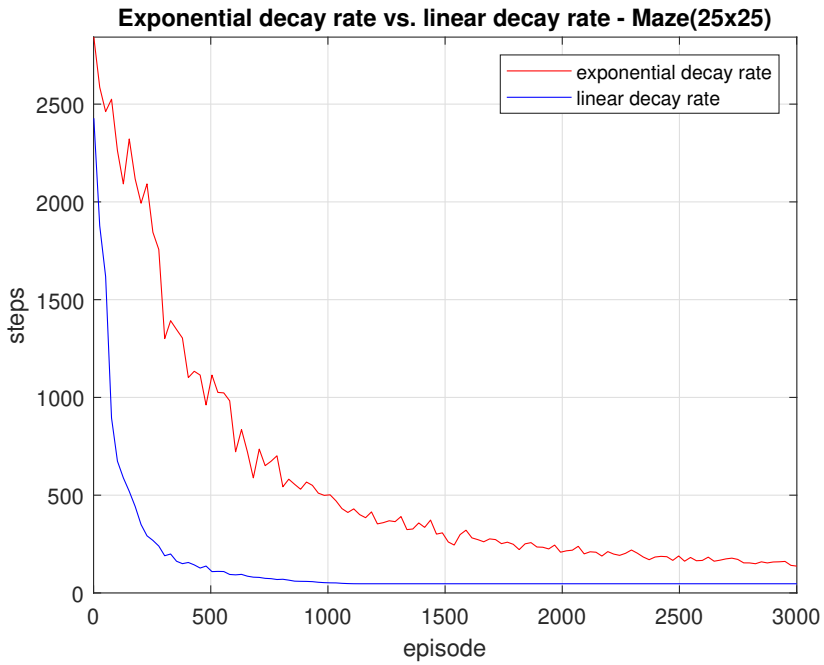

Figure 8: Exponential decay rate vs. linear decay rate - Maze 25x25

6. Conclusion

This project has shown that Reinforcement Learning in general and Q-Learning specifically are very well suited to solving the maze problem and similar games in a more general sense. In the case of a maze, especially one which the agent has never encountered before, Q-Learning can be employed in order to find a solution to the maze without the exact model of the maze needing to be provided. A maze game is easily modeled as a Markov Decision Process as the states of the game are clear and well defined and the actions that the agent has at its disposal are simple and few. A Q-table is a great choice for updating action-value pairs as the agent, in the beginning, is completely unaware of its environment (which can be represented by a Q-table initialized with 0 values). Then through a set of episodes and playing the game, getting to know the maze, the agent updates the Q-table until at some point it converges with a satisfying δ value of change.

The assignment of rewards to agent actions is also straightforward for a maze game. The agent is "punished" for every extra step and given a suitable reward for reaching the goal i.e. solving the maze. In this work moves that would land the agent on an illegal tile of the maze (a wall or outside of the maze) were disabled. An interesting alternative could be to introduce negative rewards or "punishments" for the agent, whenever it lands on a wall tile or a tile outside of the maze in order for it to learn to avoid such moves.

As could be seen from section 5. the algorithms efficiency greatly depends on the size of the maze and optimizations that would allow for better scalability to larger size mazes could be interesting for future work. Also this paper looked into exponential and linear decay of the exploration rate, which are by no means the only available options. An expansion to different kinds of exploration rate decay algorithms would also be an interesting topic for future research. As a final note, Deep Reinforcement Learning is the logical next step building on the work done in this paper. The combination of Neural Network principles with Q-Learning would allow for the agent to learn the principle of solving mazes (rather than solving a single given maze in the shortest time possible) and apply this principle to mazes it had not previously extensively explored.

References

[1] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, 2011, pp. 1143–1146.

[2] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 05 1992.

[3] M. Gunady and W. Gomaa, "Reinforcement learning generalization using state aggregation with a maze-solving problem," 03 2012, pp. 157–162.

[4] D. Osmankovic and S. Konjicija, "Implementation of q - learning algorithm for solving maze problem." 01 2011, pp. 1619–1622.

[5] M. Hacibeyoglu and A. Arslan, "Reinforcement learning accelerated with artificial neural network for maze and search problems," in *3rd International Conference on Human System Interaction*, 2010, pp. 124–127.

[6] D. Pandey and P. Pandey, "Approximate q-learning: An introduction," in *2010 second international conference on machine learning and computing.* IEEE, 2010, pp. 317–320.

[7] C. Sun, "Fundamental q-learning algorithm in finding optimal policy," in *2017 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2017, pp. 243–246.

[8] X. Yu, Y. Wu, and X. Sun, "A navigation scheme for a random maze using reinforcement learning with quadrotor vision," in *2019 18th European Control Conference (ECC)*, 2019, pp. 518–523.

[9] T. Tompa and S. Kovács, "Q-learning vs. friq-learning in the maze problem," in *2015 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, 2015, pp. 545–550.

[10] M. Marashi, A. Khalilian, and M. E. Shiri, "Automatic reward shaping in reinforcement learning using graph analysis," in *2012 2nd International eConference on Computer and Knowledge Engineering (ICCKE)*, 2012, pp. 111–116.