Mid-Term Project

January 15, 2023

Abstract

In this paper we explore various classical reinforcement learning algorithms in order to solve the OpenAI grid maze in a stochastic environment. The task combines three main challenges, each with a different maze size. For the sake of the task, the following RL approaches were used: dynamic programming, Monte Carlo, Q-learning and Sarsa. In each of the challenge we manage to solve the maze in a reasonable number of steps, considering the unstable nature of the stochastic environment. We conducted several experiments in each one of the challenges, using various hyper parameters, different ε greedy methods, rewards and value-function initializations. Furthermore, we compared the results between the different approaches. Finally, we used different incentive and deterrent rewards to try to optimize the solution of the 25X25 maze. We demonstrated That classical RL algorithms are relatively straightforward to implement in order to over come the task

1 Introduction

Reinforcement Learning is a machine learning subfield that teaches a computer program (often called an "agent") to choose the best action within his state, in order to maximize expected returns over time. The returns (and game rewards) are calculated based on the environment rules. There are two fundamental tasks in reinforcement learning: prediction and control. In prediction tasks, we are given a policy and our goal is to evaluate it by estimating the value or Q value of taking actions following this policy. In control tasks, we don't know the policy, and the goal is to find the optimal policy that allows us to collect most rewards. In this paper, we will only focus on control problems.

We differentiate between two type of RL models:

- **Model-based**: In the Markov Decision Process (MDP) world, we have a basic notion of its mechanics, meaning that we know the MDP dynamics (transition $P(s'|s,a)$ and reward function $R(s,a)$), so we can directly build a model using a famous RL equation named Bellman's equation. The final output is an "instructions matrix" (commonly called policy) composed of the best action to take for each possible state in the environment.

- **Model-free**: In Model-free learning the transition model is unknown, thus the agent relies on trial-and-error experience for setting up the optimal policy.

2 Methodology

2.1 Architecture

In order to create the maze environment, one has to import OpenAI's Gym library via supported coding language, where Python 3 was used for this article. As the default Maze has a deterministic transition model, that is, the probability of actually doing action a after choosing it is 1, we had to change it to 0.9 using a custom wrapping class. Upon initialization, the following parameters are passed to the wrapping class:

1. the environment type (e.g MazeEnvCast5x5)

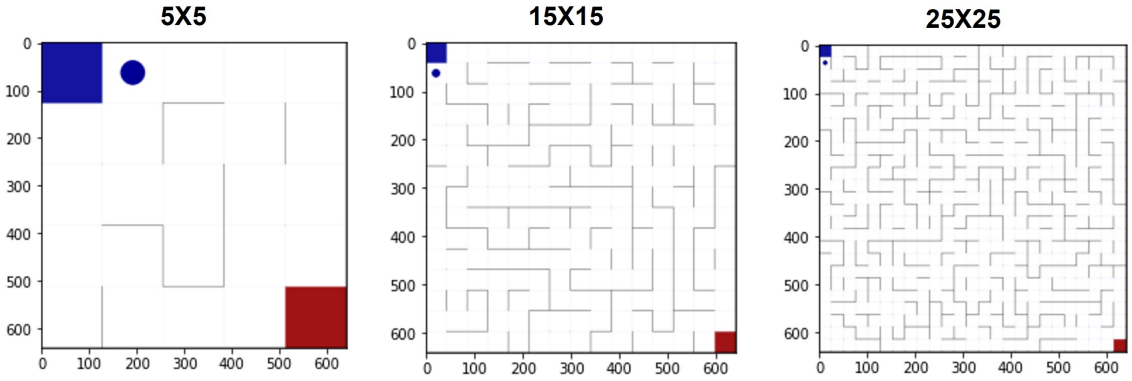
2. the stochastic probability P of the environment
3. positive reward coordinates
4. negative reward coordinates

Note that the wrapper is generic enough to handle maze variations. inside the wrapper, we customised a new rewards grid for the maze which adds positive and negative rewards in certain coordinates within the maze. Additionally, we customised the 'step' function in the following manner: each time a step is taken based on a certain action, the environment will choose this action with probability $P = 0.9$ or another random action with probability 0.1. The function will return the next state reached by the taken action, 'done' parameter which indicates if the agent has reached the end of the maze and a reward, which is based on the custom reward grid.

2.2 Experiments

For each challenge we implemented the following algorithms. For the 5X5 maze problem, a Dynamic Programming model was used, for the 15X15 maze challenge we implemented Monte Carlo, Q-learning and SARSA while the 25X25 maze problem was solved using different algorithms and techniques. Each model was provided with two videos, one during the training phase and one at the end of it, to demonstrate the training progress. At each challenge we conducted several experiments for each of the models used in that challenge and visualized the results. All three maze types are illustrated in figure 1.

Figure 1: Type Maze Challenges



3 Results

3.1 5X5 Maze

3.1.1 DP

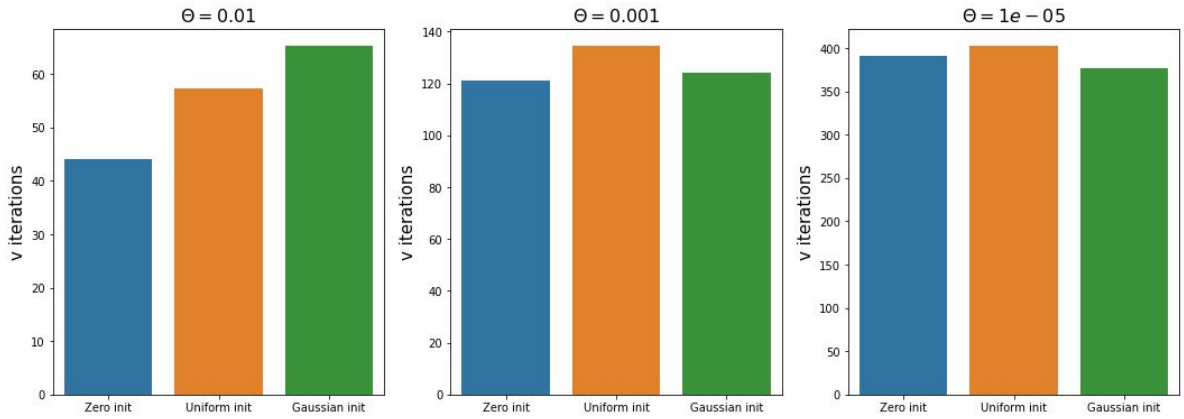
This challenge requires to solve a 5x5 Maze by implementing a Dynamic Programming algorithm. First we run the model with the following hyper parameters: $\gamma = 0.9, \theta = 0.0001$ and zero init value matrix, where θ is the convergence condition and γ is the discount factor. The model converges after 73 value function evaluations and is able to solve the maze with 8-12 steps, using the output value table. The variance in the results is due to the stochastic nature of the environment.

3.1.2 Experiments

- **V-function initializations with various θ:**

  In this experiment we try different initialization methods for the value-function. We used zero initialization $v(s_i) = 0, \forall i$, Uniform initialization $v(s_i) \sim U(0, 1), \forall i$ and standard normal distribution initialization $v(s_i) \sim \mathcal{N}(0, 1), \forall i$. Each initialization was tested using different θ values (where $\theta \in \{0.01, 0.001, 0.00001\}$) and averaged over 200 experiment runs.

Figure 2: DP - various initialization techniques and θ



As implied from figure 2, as θ decreases the number of iterations it takes for the V-function to converge increase. Furthermore, we can observe that zero initialization tends to converge faster in the first two θ values while in $\theta = 0.00001$ the standard normal initialization converge the fastest.

3.2 15X15 Maze

This challenge requires to solve a 15x15 Maze by implementing Monte Carlo, Q-learning and SARSA algorithms, which are model-free, opposed to the last section's DP algorithm, which is model-based.

3.2.1 Monte Carlo

Our custom algorithm is a mix between two common MC approaches - first-visit and every-visit methods. We start with one episode per iteration (average one instance) and continue doing so as long as the agent doesn't reach the terminal state. If he does, we update the policy and require him to do 2 first-visit episodes and average the results before updating the policy again. If he doesn't (reached step limit parameter), we update the policy immediately to make him take other paths. In general, as long as the agent reaches the terminal state, we add another episode per iteration and average all first-visits within these episodes before updating the policy again. We found that this method converges faster than every-visit and is more stable than first-visit.

The best MC model, with $\gamma = 0.999$ and $\varepsilon = 0.5$ with decay ε, converged after 330 episodes and was able to solve the maze in around 33 steps. the variance is due to the stochastic nature of the environment. Figure 3 displays the Convergence rate and cumulative rewards for the model.

3.2.2 Experiments - MC

- **Decay ε VS non-decay ε**:

  In this experiment we try two different ε greedy approaches. We run 10 experiments in which we compare the average results between MC with constant $\varepsilon = 0.1$ and MC with decaying ε with a starting value of 0.5 down untill 0.05, using the following decay formula:

  $$\varepsilon = max(\varepsilon * 0.9, 0.05)$$

  .

  As implied from figure 4, the decay method produced much better results as it took much less time to converge compare to the non decay method. Thus our ε decay method will be used in the upcoming experiments.

Figure 3: Monte Carlo - Convergence rate and Cumulative rewards

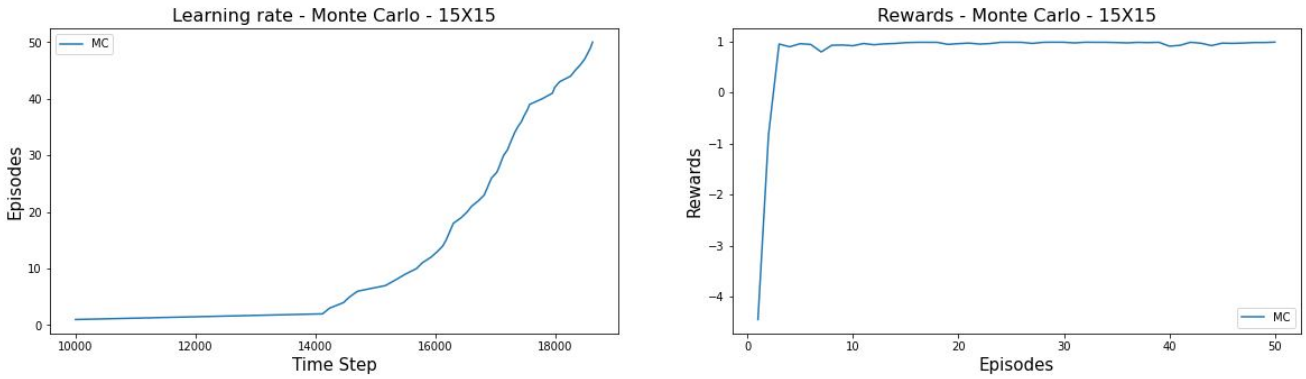
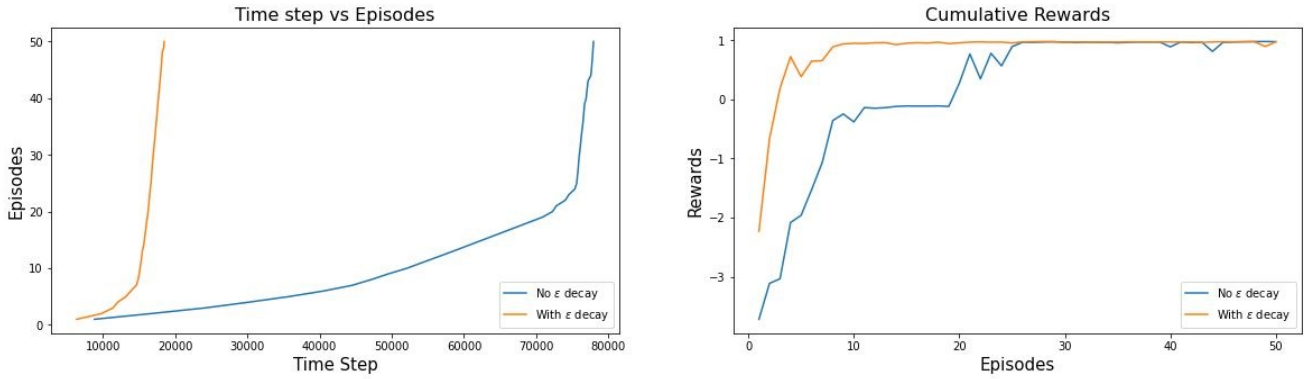


Figure 4: Monte Carlo - ε decay comparison



- **Various γ values**:

  In this experiment we try running MC with various γ values, $\gamma \in \{0.9, 0.99, 0.999, 1\}$. we run the experiment 10 times and compared the averaged results. As indicated in figure 5, $\gamma = 0.9$ didn't not converge with cumulative rewards around -1. $\gamma = 0.999$ converge the fastest, following $\gamma = 0.99$. it can be noticed that $\gamma = 1$ was lagging behind, insinuating that without discount factor the model does not differentiate well between far-way and close cell to the terminal state, making it harder to converge.

  As a side note, MC with $\gamma = 1.1$ was tested for the sake of curiosity. Obviously, the model did not converge when there is incentive to go as further as possible from the terminal state. An Overflow Error was throwed, as the state-action function was filled with very larger numbers.

3.2.3 Q-learning

The Q-learning model, with $\gamma = 0.999$, $\alpha = 0.25$ and $\varepsilon = 0.5$ with decay ε, was able to solve the maze within 500 training episodes. Figure 6 displays both Convergence rate and cumulative rewards for the model.

3.2.4 Experiments - Q-Learning

- **Hyper Parameters Grid Search**:

  This experiment was conducted in order to optimize the Q-learning model by running hyper parameters grid search for $\gamma, \varepsilon, \alpha$ and decay/non-decay ε. Each model was run for 300 episodes.

  As implied from figure 7 the dominated hyper parameter in combinations of hyper parameters is α, grouping the results into three distinct segments. The values that were used: $\alpha \in$

Figure 5: Monte Carlo - various γ values

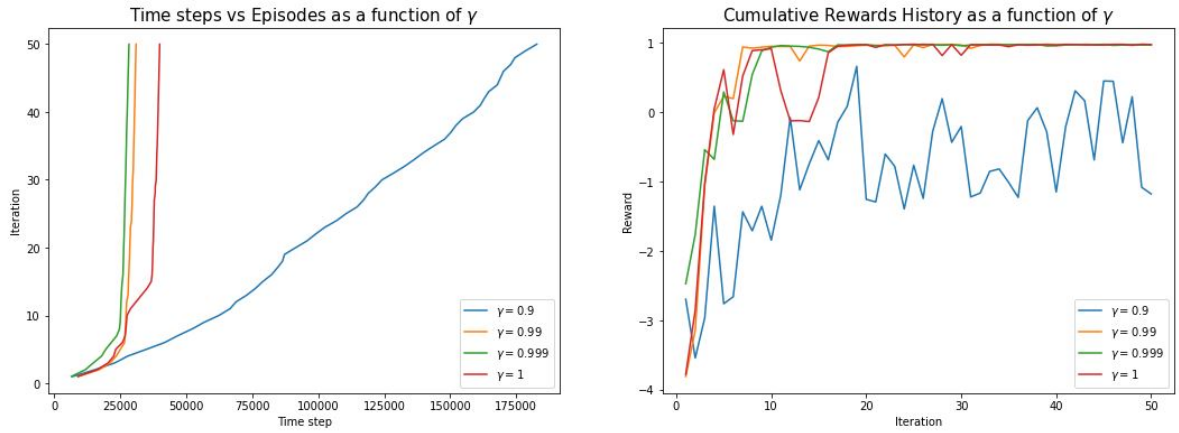


Figure 6: Q-Learning - Convergence rate and Cumulative rewards

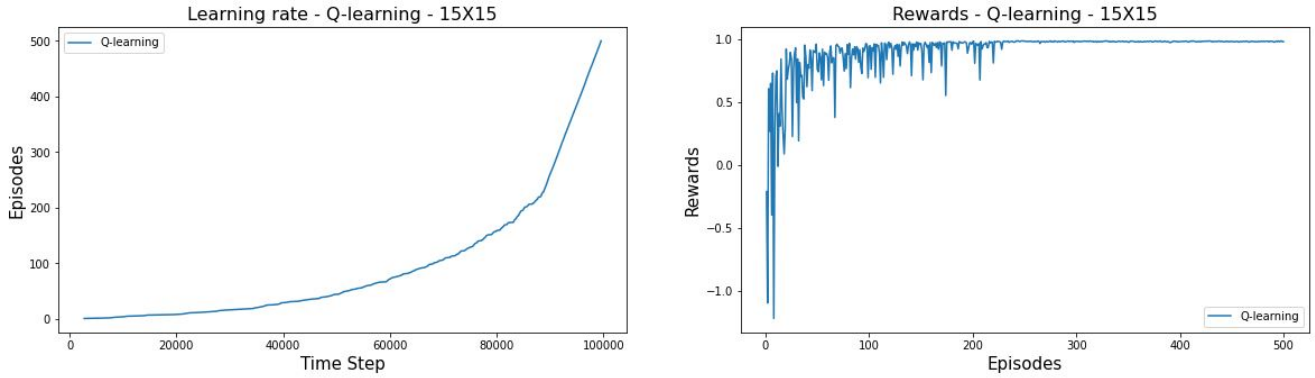
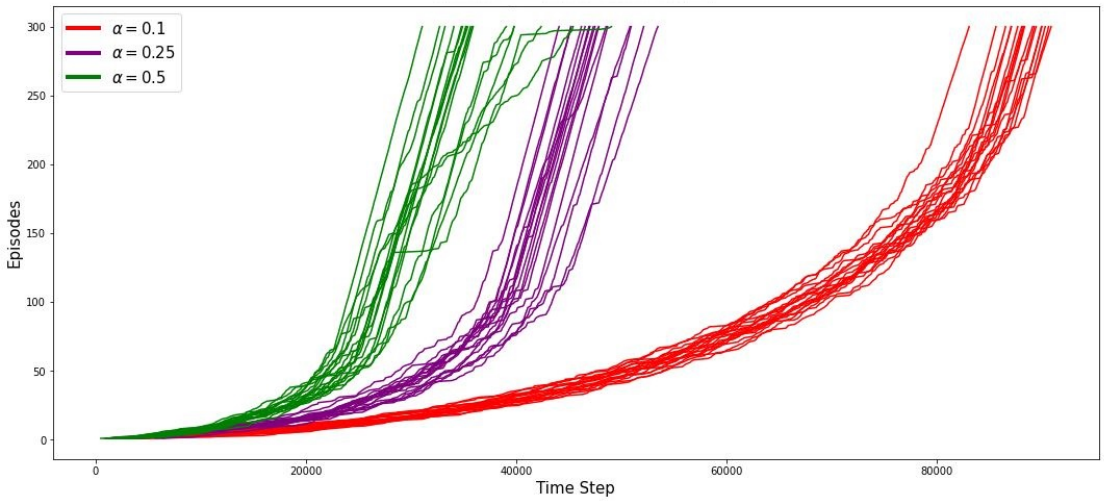


Figure 7: Q-Learning - Hyper Parameters Grid Search

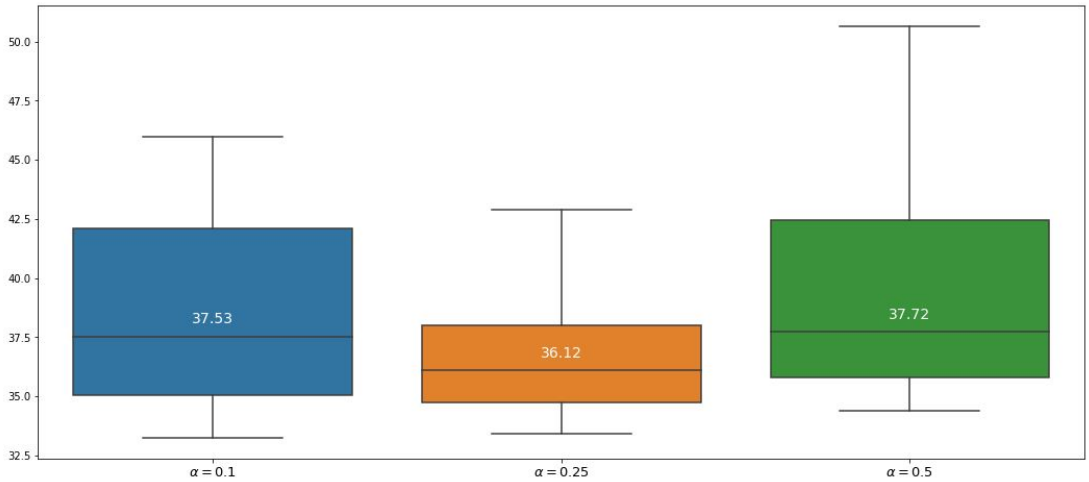


$\{0.1, 0.25, 0.5\}$, $\varepsilon \in \{0.05, 0.1, 0.2\}$, $\gamma \in \{0.9, 0.99, 1\}$, decays = [True, False].

However, this plot indicates which set of hyper parameters, grouped by α, converge the fastest in 300 episodes, but it does not provide information about which model among these models

converge to the ideal model, with the least number of steps to solve the maze. In order to address this issue, we calculated, for each α group, the average steps it took the agent to finish the maze in the last 30 episodes and visualized the results using a boxplot.

Figure 8: Q-Learning - Hyper Parameters Grid Search

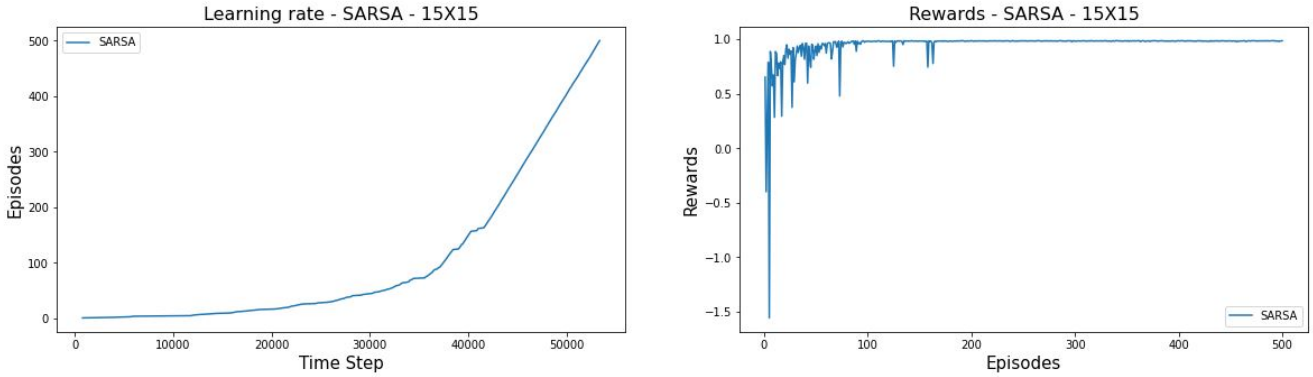


It can be observed from figure 8 that the median number of steps for $\alpha = 0.25$ was the least among the 3 α values and with lowest variance, concluding that although the models which used $\alpha = 0.5$ converged faster, they did not converge to ideal models. Using the above information, $\alpha = 0.25$ will be used for the upcoming experiments.

3.2.5 SARSA

The SARSA model, with $\gamma = 0.999$, $\alpha = 0.25$ and $\varepsilon = 0.5$ with decay ε, was able to solve the maze within 500 episodes. Figure 9 displays the Convergence rate and cumulative rewards for the model.

Figure 9: SARSA - Convergence rate and Cumulative rewards



3.2.6 Experiments - SARSA

- **Hyper Parameters Grid Search**:

  Similar to Q-Learning, a grid search was applied on SARSA algorithm as well.

  As indicated from figure 10, the results match to those produced in the Q-Learning grid search experiment. interestingly, figure 11 demonstrate the same conclusion reached for Q-Learning - that while the model converge faster with $\alpha = 0.5$, the best model is achieve by using $\alpha = 0.25$.

Figure 10: SARSA - Hyper Parameters Grid Search

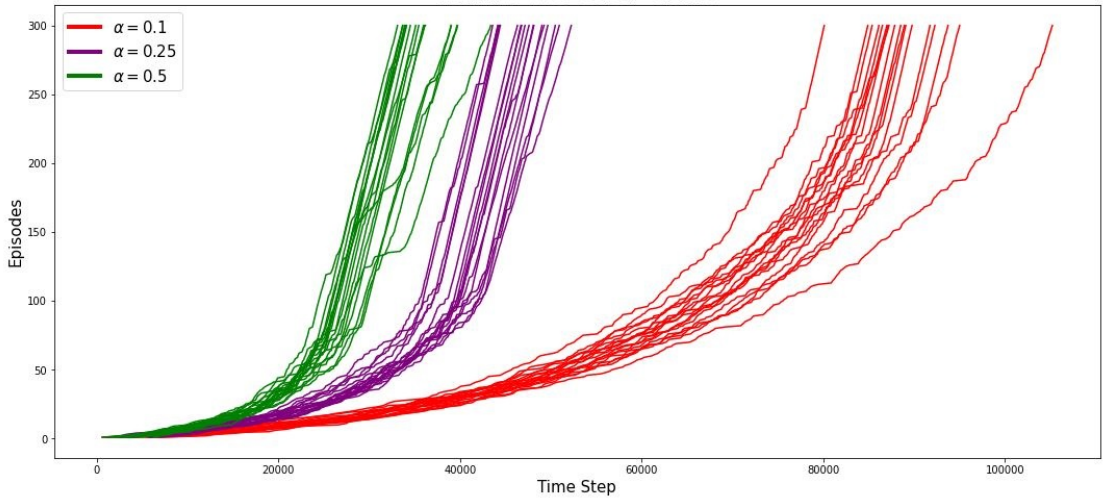
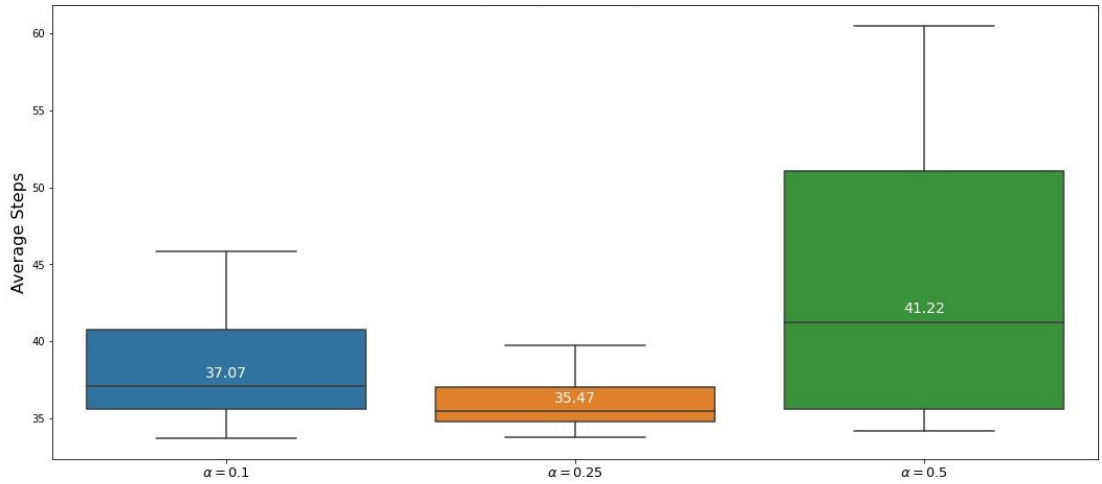
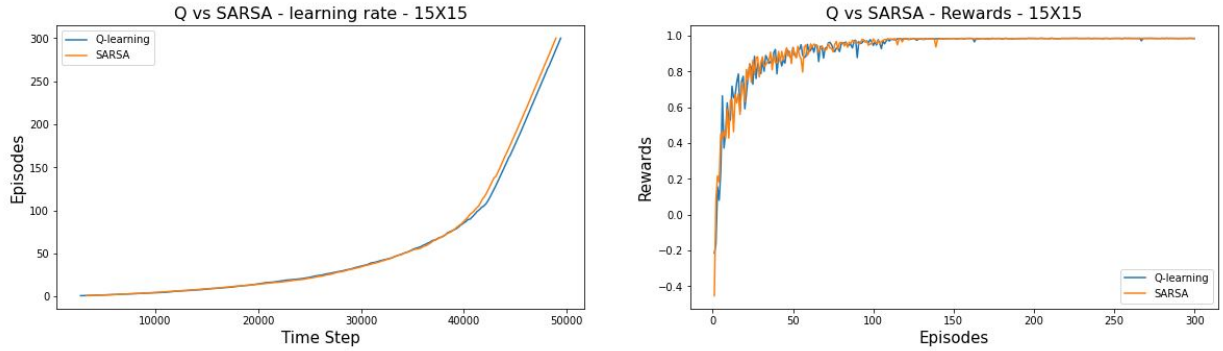


Figure 11: SARSA - Median steps to resolve the maze for each set of hyper parameters grouped by α



- **Q-Learning VS SARSA**:
  In this experiment we compare the results between SARSA and Q-Learning.

Figure 12: Q-Learning VS SARSA



As implied from figure 12, SARSA converge a bit faster than Q-Learning with $\alpha = 0.25$, $\gamma = 0.99$ and ε with ε decay.

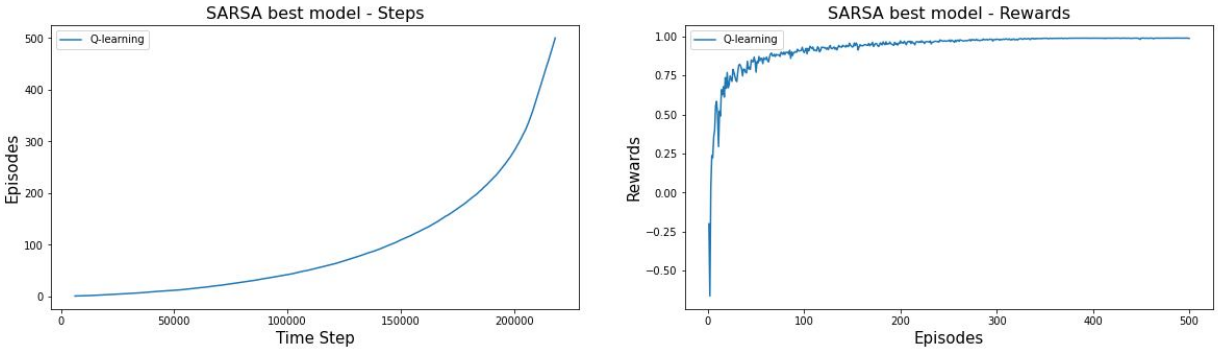
3.3 25X25 Maze

Previously we compared between SARSA and Q-learning while doing research for the optimal set of hyper parameters. We decided to choose SARSA over Q-Learning as the model-free model to handle this Challenge, as it produced slightly better results than Q-learning. Furthermore, we also wanted to compare between Model-Free and Model-based algorithms. Thus, we compared running time between dynamic programming and SARSA.

- **SARSA - Convergence rate and Cumulative awards**:

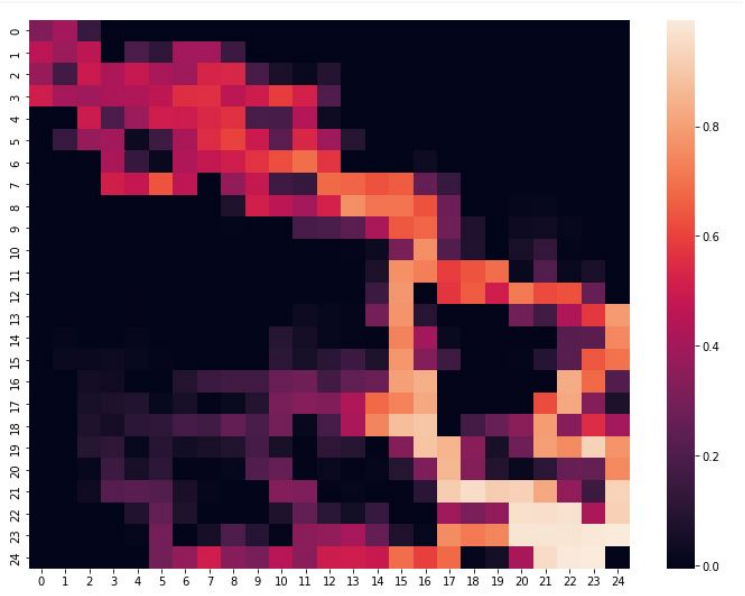
  The SARSA model, with $\gamma = 0.999$, $\alpha = 0.25$ and $\varepsilon = 0.5$ with decay ε, was able to solve the maze within 500 episodes. The results are composed of 10 runs average. The results of the Convergence rate and cumulative rewards is shown in figure 13

Figure 13: SARSA - Convergence rate and Cumulative awards - 25X25 maze



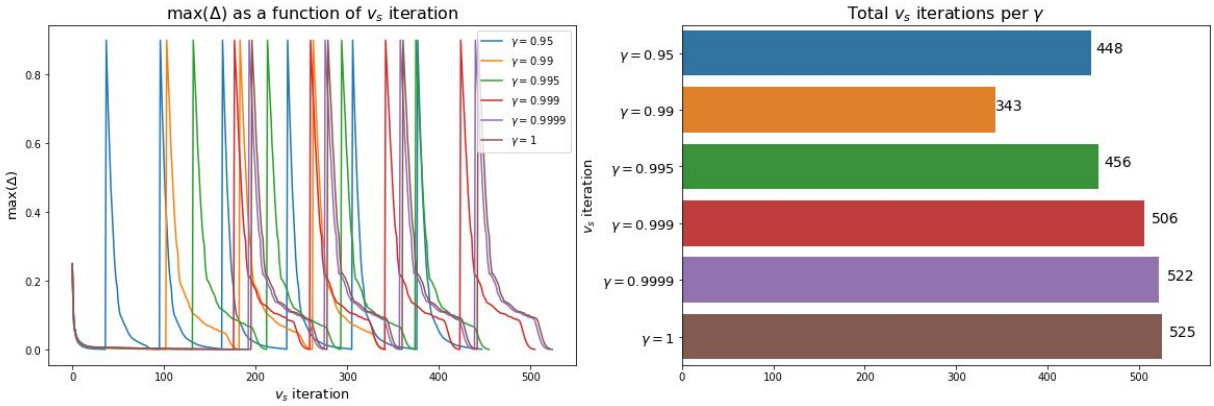
- **SARSA - State-value Heatmap**: In order to visualize the agent's path towards the Terminal state in the SARSA model, we average the state-action function across the 10 experiments and choose the max state-action value for each state. The following heatmap in figure 14 provides a nice representation of the average agent's path to the Terminal state.

Figure 14: SARSA - State-Action function heatmap



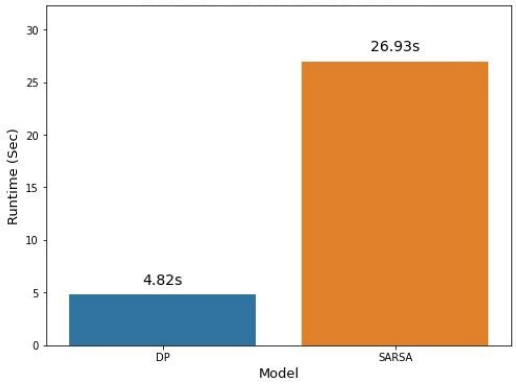
- **DP- model efficiency as function of γ**: In this experiment we run DP with various values of γ where $\gamma \in \{0.95, 0.99, 0.999, 0.99991\}$. In each run we summed up the number of value-function evaluations till the model converged. Furthermore, we documented Δ, which is the maximum difference between the new value-function and the previous one. Its values were used to demonstrate the convergence of DP in each policy evaluation. The results are shown in figure 15. excluding $\gamma = 0.99$ it can be noticed that there is a positive correlation between γ values and the number of value-function evaluations it took the model to converge. in addition, all the models behave similarly in terms of convergence in each policy evaluation, however in each model it took different number of value-function evaluations to complete a policy evaluation. For all values of γ except $\gamma = 0.99$ it took four policy iterations for the model to converge, while with $\gamma = 0.99$ it took only three. Thus with less policy iterations and with the least value-function evaluations, we picked $\gamma = 0.99$ to use when running DP on this challenge.

Figure 15: DP - Convergence rate as a function of Gamma



- **DP VS SARSA**: We ran SARSA 10 times, averaged the running time and compared it with the time it took DP to complete the policy iteration process. The result is shown in figure 16. SARSA's running time was around 5.5 times slower than DP's.

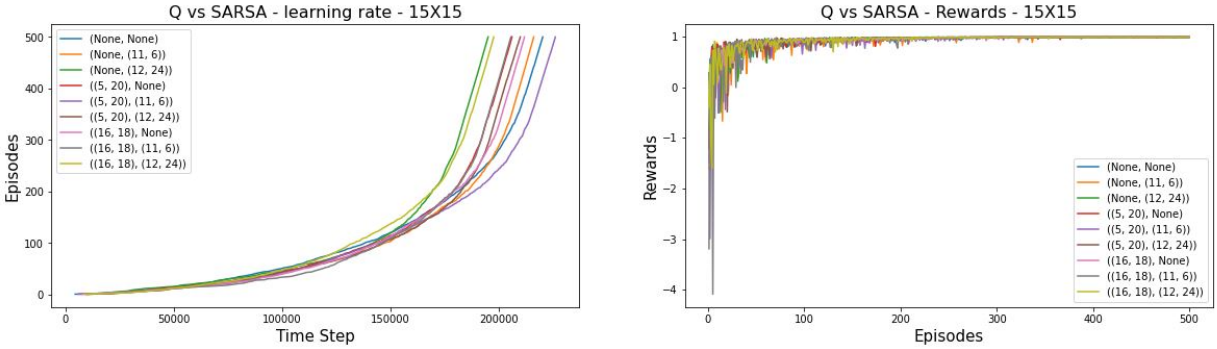
Figure 16: DP VS SARSA - Running time - 25X25



- **Positive and Negative Rewards** In this experiment we tried to optimize our SARSA by using positive and negative rewards in different locations, some with only negative/positive rewards. The coordinates were chosen using a poll, conducted by one of the authors in his workplace. The poll introduced the maze problem and each questioneer was asked to give both positive and negative reward coordinates to help the agent solve the maze. It is important to mention that this method implements "imitation learning", which means that the agent's initial decision making

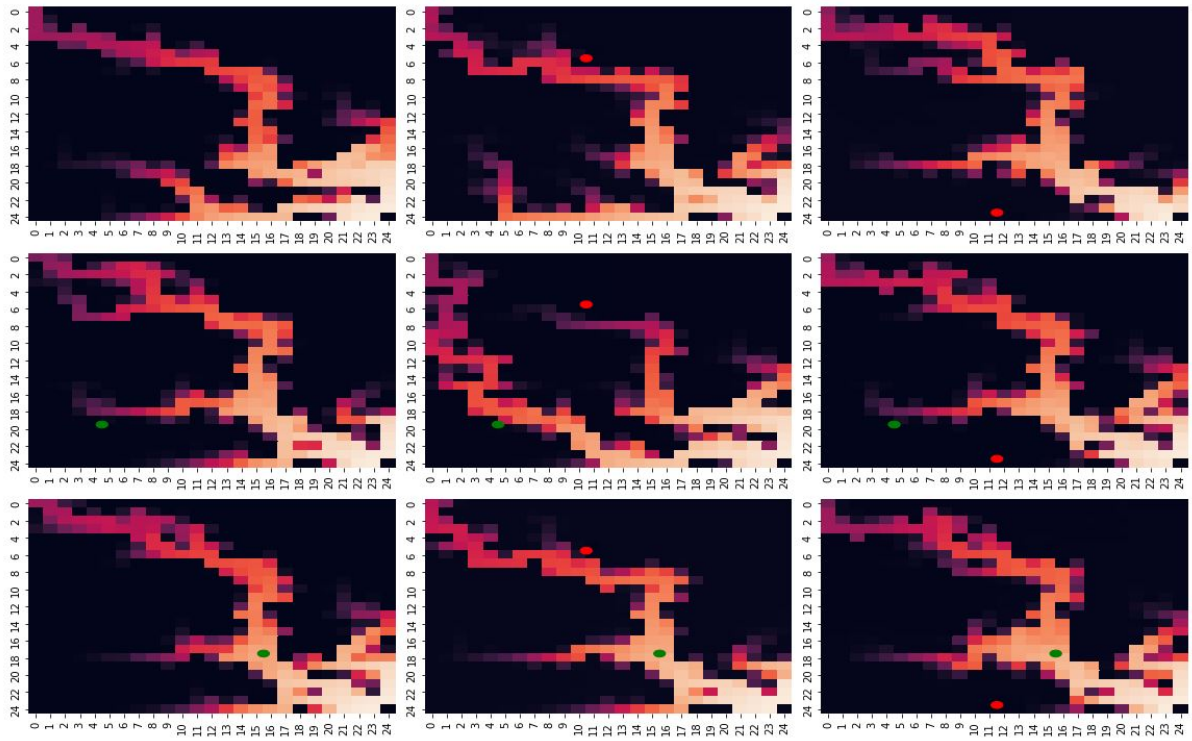
is influenced by the help of humans. Using the poll, total of nine location variations were used. We resort to the Heatmap shown in 14 to get a notion where to position these rewards in the reward grid. As implied from 17 several combinations of positive and negative rewards locations improve the convergence rate of the model. The fastest model had only a negative reward located in $(12, 24)$ following a model with positive reward positioned at $(16, 18)$ and negative reward at positioned at $(12, 24)$. The vanilla model, without positive and negative rewards, was second to last in its performance. Finally, figure 18 portray the various Heatmap produced by these

Figure 17: SARSA - Convergence rate and Cumulative rewards - positive and negative rewards locations



model. The green circles represent positive reward, while red circles represent negative rewards. The fastest model to converge is located on the top right corner of the figure, while the original model is on the top left. Focusing on the centered Heatmap, it may be noticed that the specific pair of positive and negative reward locations used in this model, were able to change the agent path to a completely different route than the original model.

Figure 18: SARSA - State-Action function Heatmap - positive and negative rewards locations



4 Conclusions

In this project we reviewed several classical reinforcement algorithms in order to solve OpenAI's maze challenge. While experimenting with the various models, we observed high variance in the results when changing certain hyper parameters. This is specially relevant to the model-free algorithms, as switching between decay and non decay ε or changing α had considerable affect on the outcome. Furthermore, we noticed that SARSA provides slightly better results than Q-Learning in terms of convergence time, which might suggest that SARSA preforms better than Q-Learning in a stochastic environment. Finally, We observed that the dynamic programming model overcomes SARSA in the 25X25 maze challenge, with a considerable run-time advantage. Thus, DP might still serve as the preferred tool for simple environments.