RUNI: Reinforcement Learning

MidTerm Project RL Maze Report

Fall Semester 2022-23 TASHPAG

Submitted by:	**Yarden Fogel ID: 011996279 (yarden.fogel@post.runi.ac.il)**
Partner:	Sharon Koubi ID: 301315040 (sharon.koubi@post.runi.ac.il)
Date:	January 16, 2023

Maze Game Academic Paper

1 Introduction

This project solves a stochastic 2D maze with 3 environment configurations ($5x5$, $15x15$, and $25x25$) and employing the use of various flavors and combinations of four reinforcement learning methods: Dynamic Programming, Monte-Carlo, SARSA, and Q-Learning.

The maze environment has an observation space of NxN and allows the agent to move in 4 directions: Up, Down, Left, and Right. Whenever the agent chooses an action, the actual step taken can differ from the chosen direction with a probability of $p = 0.1$, thus converting what was initially a deterministic action into stochastic probabilities. The agent however is blind to this change from deterministic to stochastic, and tries to learn the best policy with minimal effort (and Colab GPU units!), according to the state and rewards resulting from the action chosen.

Our approach overall was to create a training, testing, and tuning-optimization framework that could handle a wide array of environments, parameters, and exploration and/or exploitation tactics and strategies. Our aim was to mobilize this flexible architecture to optimize training time and convergence, and to do this in a clean, easy-to-follow, intuitive manner. We hope that comes across in the mountain of code we link below, and in our report that follows.

2 Description of Our Mission - OpeRation MazeL

2.1 $5x5$ Maze

We initially tackle a $5x5$ 2D maze, and we solve with policy iteration which is informed by the use of Dynamic Programming to determine the V or value of different states.

2.2 $15x15$ Maze

We are then faced with solving a large $15x15$ 2D maze environment, which has an observation space of 225, or $9x$ larger than the initial $5x5$ configuration, and we attempt to solve this task with 3 approaches that all entail the agent recording its experience and leveraging that to improve as it goes along - Monte Carlo, SARSA, and Q-Learning. Throughout this section and the last section (described below), we experiment with a range of hyperparameters and initializations in an effort to optimize training speed and performance / convergence speed for the agent after the training phase.

2.3 $25x25$ Maze

Finally, in the last part of this project, we are presented with an even larger environment of $25x25$ and an observation space = 625, or $25x$ larger than our original little $5x5$ maze. Here

we have our pick of the litter for which approach and methodologies we use to optimally solve this larger maze in a competitive way compared to our peers in the class. We will describe our process for choosing and testing which approach to tackle this problem with in detail in the sections that follow. Of note in this section is that we were also blessed with additional degrees of freedom in that we could include positive and negative rewards at locations of our choosing, to further assist our beloved agent towards an optimal and hopefully fiercely competitive solution and convergence path.

3 Methodologies and Design of Our Maze-Solving Machine

3.1 General Approach and Motivation

As mentioned above, we aimed to construct and employ an agile, efficient agent optimization machine that can handle anything thrown its way, thoroughly test and demonstrate experimentation along the way, and ultimately reach the optimal outcomes quickly and for the right reasons.

3.2 General Design Concepts - Environment Preparation

3.2.1 Environment Wrapper - Stochastic Steps

The notebook provided contained a deterministic environment named "MazeEnv", and we addressed this by adding a Environment Wrapper classes - in the 'EnvMetadata' class, the '_get_transition_model' method effectively teleports the agent into an alternate stochastic universe. The 'step' method inside the 'EnvWrapper' class then translates that into actual moves traversed by the agent, factoring in the action chosen and the adjustment for the stochastic probabilities of going somewhere else.

3.2.2 Creating Robust Environment and Overcoming Failures

Additionally, in order to handle the crash mentioned in the initially provided notebook - that if we run the environment setup and it breaks / crashes, to simply try again and rerun - we implemented a 'env_failed_workaround' method. This workaround method reruns the environment creation process until it works properly, and creates a robust foundation - which ensures that it will work without failing - and we were then able to utilize this construct throughout the entire exercise.

3.3 Other Starting Configurations

The default starting point is always the $(0, 0)$ cell at the upper left corner, and the ending point is the $(N-1, N-1)$ cell at the lower right corner of the maze. The original rewards are $\frac{-0.1}{number of steps}$ for any step other than the goal or terminal cell, which has a reward of $+1.0$.

3.4 Epsilon-Greedy and Epsilon Decay

The agent uses an **epsilon-greedy** approach for exploration-exploitation of the state-action space. We found that when the epsilon ϵ is 'too low', i.e. close to or approaching 0, then the agent doesn't do enough (or any) exploration, and the agent doesn't converge. So clearly some exploration is critical to the agent's learning process. If the epsilon was 'too high', then the agent almost won't move at all without any randomization, further demonstrating how the chosen ϵ and how it decayed throughout the episodes really dictated the number of steps it took the agent in each episode.

While we tested various other approaches discussed in greater detail later in the report, our backbone epsilon decay methodology was based on the following simple linear decay factoring:

- Given an episode n out of a total of m episodes, the decay factor was $\frac{(m-n}{m}$ and we multiplied this decay factor by the initial epsilon ϵ. For example, in the first episode, $n = 0$ and $m = 500$, so the decay factor is $\frac{500-0}{500} = 1$, so it starts with our initialized epsilon ϵ. At the last iteration, $n = 499$ and $m = 500$, the decay factor of $\frac{500-499}{500}$ approaches 0, and thus our epsilon ϵ at that stage will also approach 0.

3.5 Training, Tuning and Experimentation Constructs

3.5.1 Training

Since our agent is always training - be it in a $5x5$ or $25x25$ maze, with DP or MC, changing policies, testing and tuning hyper-parameters and initializations, etc - we were careful to consider training efficiency and speed to convergence in all our solutions, perhaps even to a fault.

3.5.2 Training and Tuning Time-Out Limit and Implications

When tuning hyper-parameters, initializations, and policy approaches, we set a 2-minute run-time limit, whereby the training for that episode will time out and move on. The set of possibilities and permutations for testing and fine-tuning is effectively infinite, so we wanted to create a hard cut-off threshold that allows us to experiment, and for the agent to explore, but that still delivers an efficient experience to the user running and testing our framework. This 2-minute timeout was the balance we found most appropriate, as we noticed repeatedly that when the time exceeded that, it almost always coincided with the agent getting 'lost', going in endless loops, not converging, or at best, not gaining sufficient incremental information to warrant continuing beyond that hard limit. To be clear, our decision here is not perfect for all cases, and while it facilitates fine-tuning, it factually reduces the explorative runway the agent has to work with, and it's certainly possible that this could have created blind spots or missed opportunities to gain really meaningful insights in such edge cases...food for thought.

3.5.3 Experimentation Framework for Tuning Methods, Inits and Parameters

In order to search out the optimal parameters, policies, assumptions, and initializations, we developed experiment class handlers, including class ExpParameters - for populating parameters and easy decorating and printing of params being tested, and 'Experiment Auxiliaries' - which included several simple functions for generating and displaying graphs and charts.

Our experiments then return the optimal values for the parameters tested - which we defined as those which minimize the total number of training steps/time, and converge the fastest to max reward expectations.

3.6 Technical Challenges

Below we provide brief commentary on the various aspects of our design and implementation, methodologies, 'KPIs', and challenges faced.

- Platform / Technical Challenge 1: We ultimately used Google Colab Pro, after attempting on local dedicated machines, IDEs, local notebooks, etc. This project did not really lend itself to running with local compute. We were able to get it to run on a dedicated local machine, but it was not repeatable or reliable and extremely slow. The limitations of the freemium Colab were resulted in losing work and lots of time and redundant run-time - so we ultimately felt compelled to surrender and paid for a Colab Pro account.

- Primary Technical Challenges 2 - Game_Break: We also ran into the model 'breaking' several times, where the game just blew up. There were also some infinite loops and the agent getting 'lost' and not converging, but beyond that we also ran into cases where the entire game environment collapsed. However, we were able to overcome this with our env_failed_workaround described above.

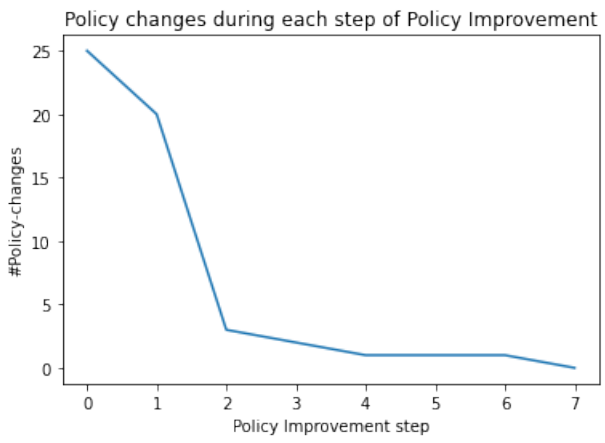
4 Environment-Specific Implementations and Experimental Results

A note regarding our presentation of methodologies for each environment and the results of our experiments - we will provide some graphics, but not all, and instead focus the report on the key metrics and learnings. An almost overwhelming wealth of visuals are available within our notebook.

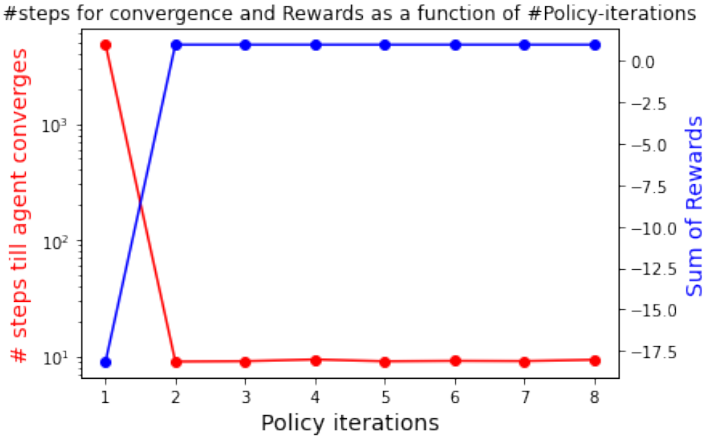
4.1 5x5 Maze Environment - Dynamic Programming:

The Dynamic Programming approach to populate the table of values V for the various states of our environment action-space does so by evaluating the action-space set in its entirety and plans a solution to solve that accordingly. However, doing so involves an implicit assumption that there exists a known transition model, and one that can demonstrates all the possible states that can be reached by the agent as a result of taking the possible actions from the current state. These assumptions and requirements may often not even be possible in more demanding real-life use cases, and if it is possible, then it will often by impractical and computationally tough to justify. We now present some graphics demonstrating our solution to the $5x5$ maze.

The first chart below demonstrates the algorithm's convergence by showing the reduction in policy changes, until it reached 0 policy changes, during the policy improvement step.
We then calculate the reward as a function of the previous policy during the learning and training phase, which we show in the below chart, with the number of steps require until convergence and the sum of rewards, as a function of number of policy iterations.



(a) DP Reduction in Policy Changes



(b) DP Steps/Reward as Function of Policy Iters

Figure 1: DP Convergence

As we can see in the above graphic, the algorithm more or less converges after the first cycle of policy iteration, the reward reached its maximum possible value, and the number of steps required for convergence became constant thereafter.

In the two graphs that follow, we test the rewards and number of steps until convergence as a function of our Θ or threshold size parameter.

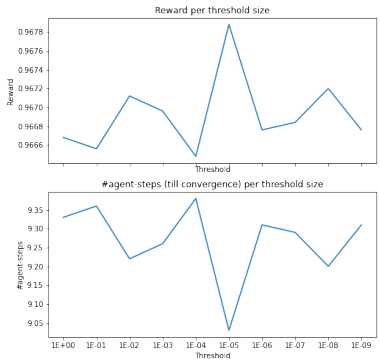


Figure 2: DP Reward and Number of Steps as Function of Threshold (Θ)

As we can clearly see from the above 2 charts, the best reward was achieved with $\Theta = 1 \exp -5$, so we stay with this value and use it as our optimal threshold.

4.2 15x15 Maze Environment - Monte Carlo:

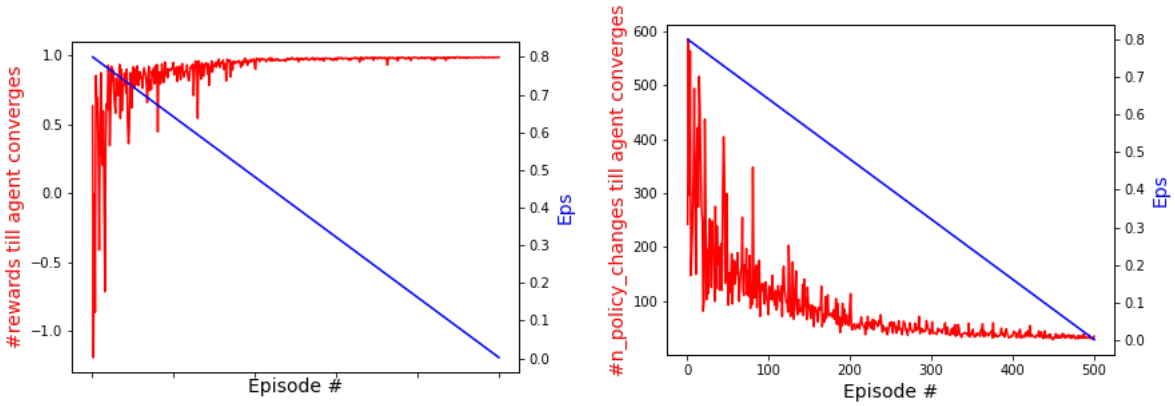
The following methods used in the $15x15$ environment (Monte Carlo, SARSA, and Q-Learning) all involve the agent learning from its trial and error and using that information to try to continually improve. Unlike with the Dynamic Programming approach above, these three methods do not require assuming a transition model, thus rendering them potentially more useful and applicable to solving other problems beyond the scope of this project, and much more so than is the case with the Dynamic Programming approach.

4.2.1 Monte Carlo Summary Results

- Steps at 200, 250, and 500 episodes $134, 29, 38$
- Rewards at 200, 250, and 500 episodes $0.9409, 0.9876, 0.9836$

Commentary: As we can see in figure 3 where we have the MC convergence plots, the reward converges to 1, as the number of episodes gets bigger. We can see that it could have been stopped after 200 episodes since that no significant change in reward achieved after that point in time.

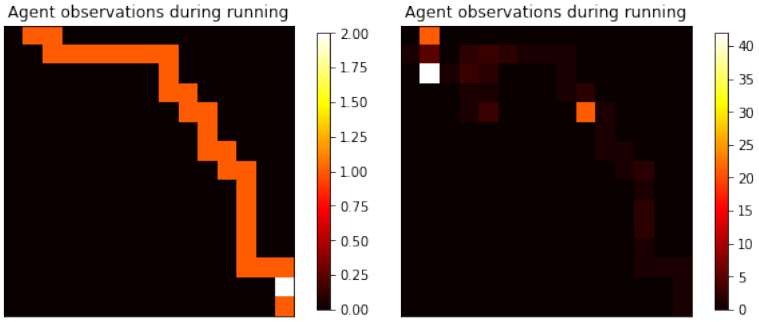
In Figure 4, we show the heatmap trail of the agent throughout the course of training. Since the algorithm converged after $2\tilde{5}0$ steps, we cannot demonstrate the improvement based on this policy and we need to choose a policy from an earlier episode, so we show episode 200 in the second map (b). It's also clear from the above stats and the heatmap trails that the mid-point at 250 episodes was more optimal than at 500, as there is over-fitting and over-exploration once we train on 500 episodes compared to 250.



(a) Rewards

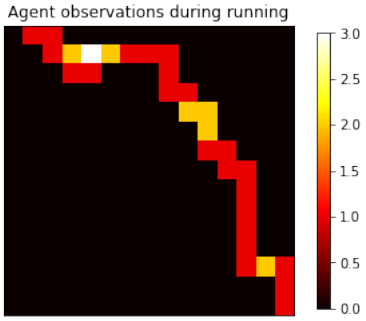
(b) Policy Changes

Figure 3: MC Convergence



(a) Middle of Training

(b) At 200 Episodes

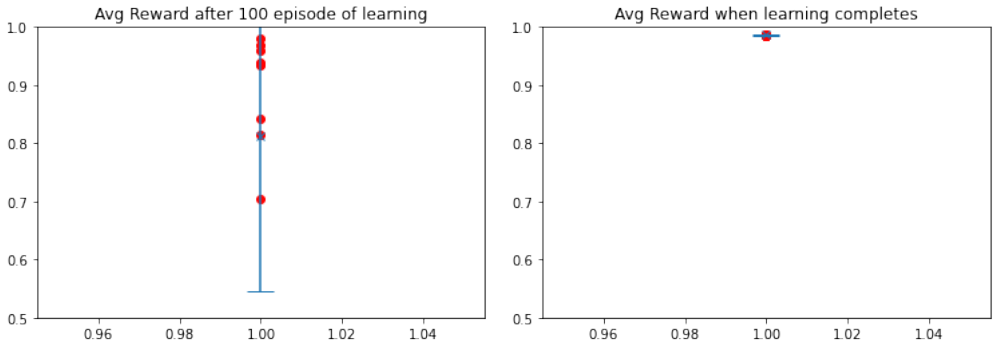


(c) End of Training

Figure 4: MC Training Path - middle to end

4.2.2 Monte Carlo Average Rewards

We took the policy after 100 episodes of training, and then take the average of 10 agent iterations thereafter, which as we see in the first part of **Figure 5**, is still very noisy, demonstrated by the large standard deviation (and can take a long time to run). The second average reward chart in figure 5 shows the same experiment after training completes.



(a) At 100 Episodes

(b) End of Training

Figure 5: MC Frozen Policy - Average Reward

4.2.3 Monte Carlo Testing and Tuning

In the below graphs, we demonstrate some of our parameter and assumption tuning for the Monte Carlo approach to solving the $15x15$ maze.

epsilon decay function testing:
The exponential decay was awesome! No-decay makes noisy convergence as expected with $\epsilon = 0.8$ (exploration mode). We have so many other tests that aren't shown here but we provide the summaries below, and many other charts and tests available inside the notebook.

4.2.4 Monte-Carlo Tuning - Best Parameters

We then also run the MC model with the best parameters, which clearly shows the tendency of cherry-picking to cause some overfitting.

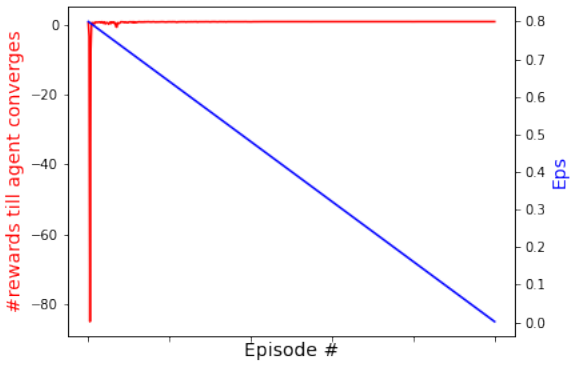
4.2.5 Monte Carlo Optimal Parameters

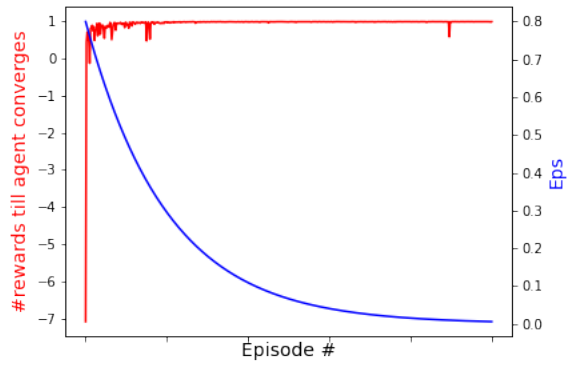
- π = "Go-to-target" , Q = all zeros, $\gamma = 0.9$
- $\epsilon = 0.8$, ϵ decay = exponential, first vs. every-visit = first visit

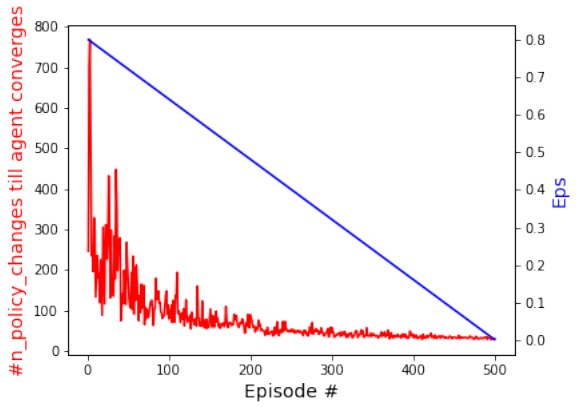
4.2.6 Monte-Carlo Tuning - Final Thoughts

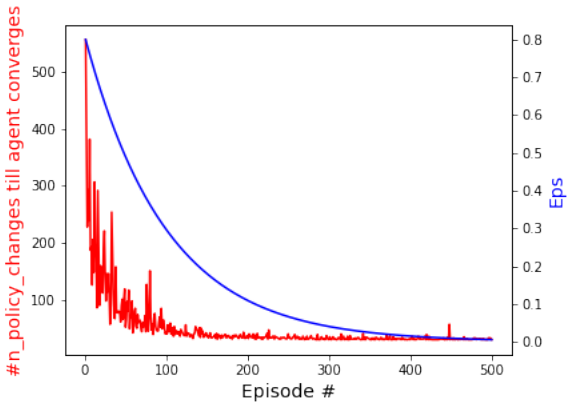
Note: When we tried an initial policy that does not allow the agent to go in some direction, for example go always up, the agent couldn't find the path to the terminal state. Therefore, the algorithm in such a case can't reach convergence. When we used the random initialization (first experiment), the algorithm converged slowly after $\tilde{150}$ episodes.

Surprisingly, when trying to set a policy with direction i.e. "go-to-terminal-direction", the algorithm converged quickly (after only few episodes). This policy means that the agent can go Down/Right with probability of $p = 0.4$, and Left/UP with probability of $p = 0.1$. The reason for this behavior is probably due to the fact that when the agent is pushed throughout towards the terminal point, even if it doesn't go exactly that way, this push can assist the agent in avoiding obstacles along the way. And the general momentum pushes it to the terminal state.



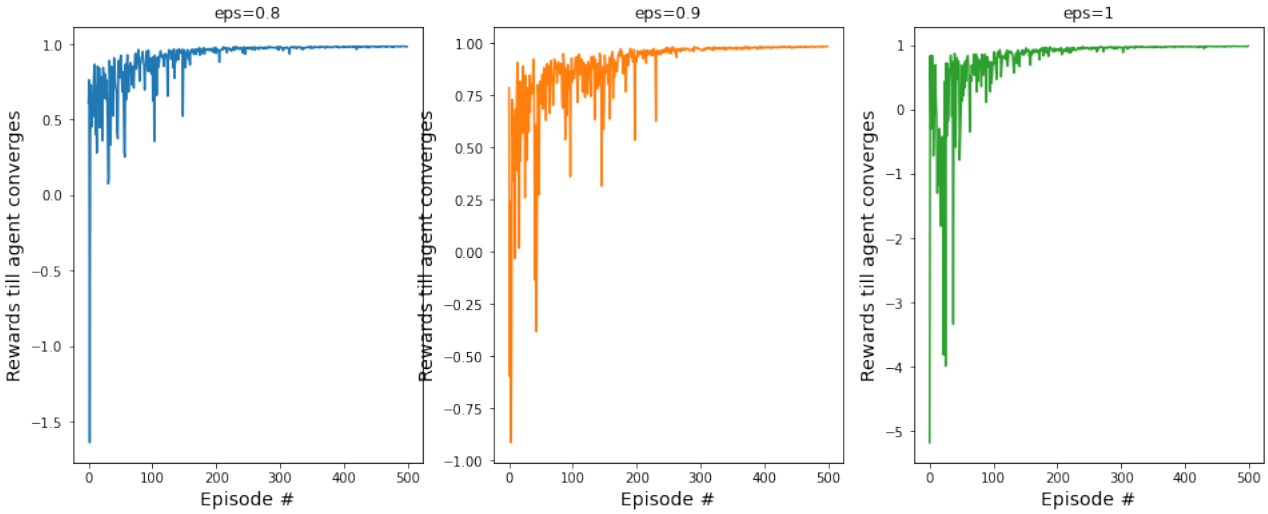






(a) epsilon decay - linear

(b) epsilon decay - exponential



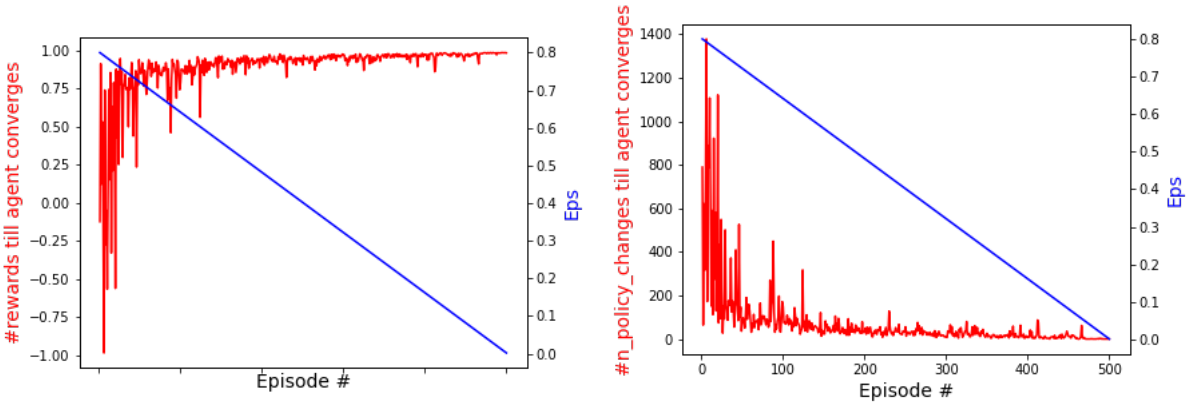
(c) epsilon soft

Figure 6: MC Parameter Tuning - Epsilon

4.3 15x15 Maze Environment - SARSA:

4.3.1 SARSA Summary Results

- Steps at 250, and 500 episodes $591, 30$
- Rewards at 250 and 500 episodes $0.7378, 0.9871$



(a) SARSA Convergence - Rewards

(b) SARSA Convergence - Policy Chg

Figure 7: SARSA Convergence

4.3.2 SARSA Testing and Tuning

Q Inits:
Random $(-1)to(-5)$ and "all -1" both didn't converge. It is probably since the agent was pushed into some areas with greater reward (-1) and didn't reach the terminal state in a reasonable amount of time to be able to understand that it has some better reward waiting at the end. You can see it in the provided heatmaps of "Random $(-1)to(-5)$"

ϵ **decay functions:** Again, the exponential behaved smoothly rather than the other decay functions.

α **:** With $\alpha = 0.3$ we're getting the most stable learning process, while it still allows for enough valuable exploration time at the beginning of training. For $\alpha = 1$ the agent didn't have time for exploring and when it got stuck, it didn't know how to get away from that problem trap region.

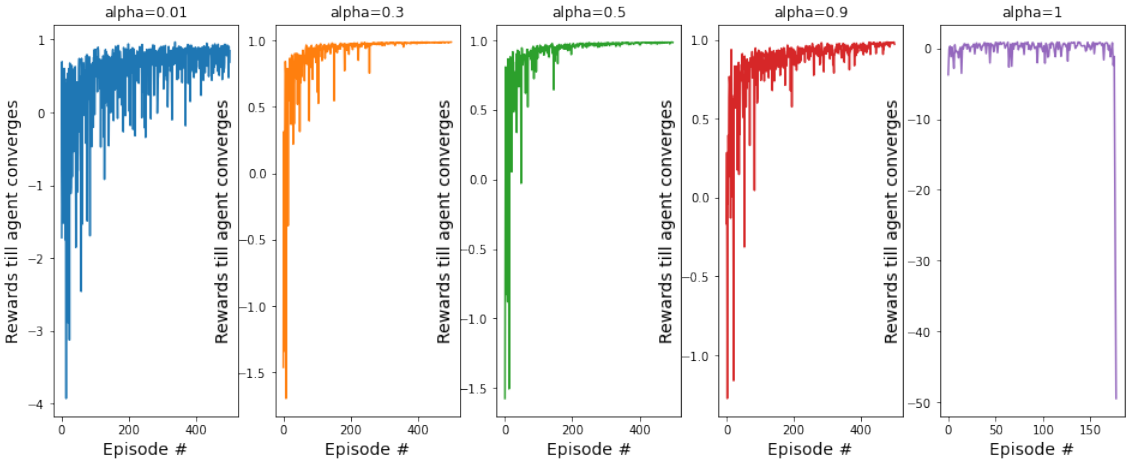


Figure 8: SARSA α Tuning

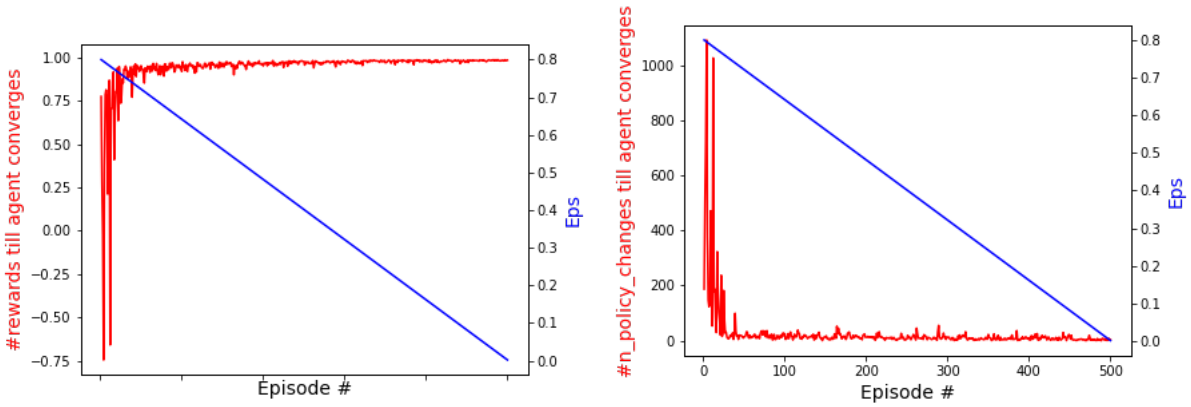
4.3.3 SARSA Optimal Parameters

- π = random, Q = all zeros, $\gamma = 0.5$
- $\epsilon = 0.9$, ϵ decay = exponential, $\alpha = 0.3$

4.4 15x15 Maze Environment - Q-Learning:

4.4.1 Q-Learning Summary Results

- Steps at 250, and 500 episodes $78, 53$
- Rewards at 250 and 500 episodes $0.9658, 0.9769$



(a) Q-Learning Conv - Rewards

(b) Q-Learning Conv

Figure 9: Q-Learning Convergence Plots

4.4.2 Q-Learning Testing and Tuning

Q Inits:
Same exact behavior like SARSA with the same explanation. Make sense since both had the same cold start. See how the agent follows after peeks of (-1) in the world:

γ **experimentation:** Following to the previous results, it seems that as long as the algorithm converges, the jittering around is good for us, in that it allows the agent to explore the world around it and learn more areas of the map and crucial information relating the connections between all the areas and paths travelled. Therefore, to allow for this, we'll go with $\gamma = 0.3$ as an optimal choice for the Q-Learning algorithm in this $15x15$ environment.

ϵ**:** $\epsilon = 0.8$ has a good balance between exploration (jittery at the beginning) and exploitation (convergence at the end).

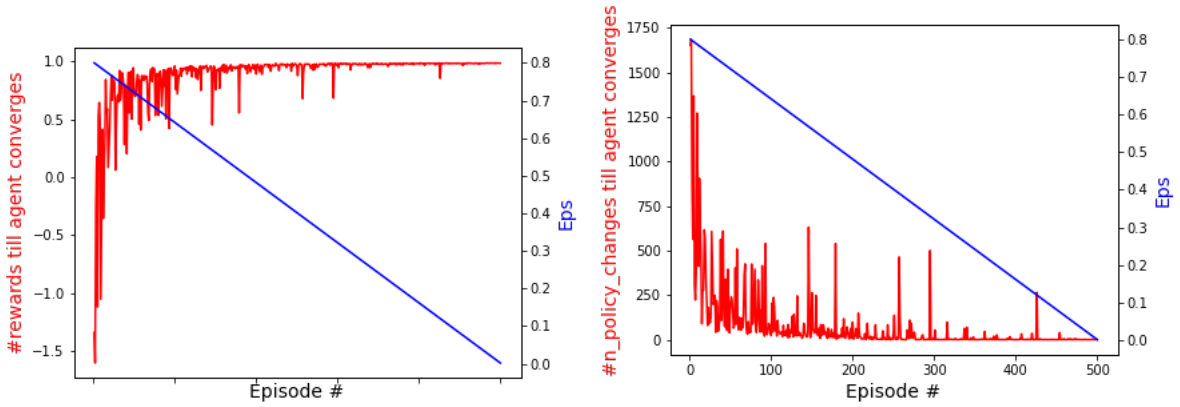
ϵ **decay functions:** Again, the exponential behaved smoothly compared than the other decay functions. We'll go with the linear function since it allows more time at exploration and the algorithm converges at the end.

α **:** With $\alpha = 0.2$ we're getting the most stable learning process, while we still have enough exploration time at the beginning. As might be apparent throughout the exercises, striking this balance between exploration and exploitation - to stumble into and gain valuable information - while also racing to the most efficient and maximal destination - is critical to our thought process and strategies, and foundational to our entire approach.

4.4.3 Q-Learning Optimal Parameters

- π = random, Q = all zeros, $\gamma = 0.3$
- $\epsilon = 0.8$, ϵ decay = linear , $\alpha = 0.2$

OFF-POLICY TD CONTROL - Tuning Q-Learning algorithm with best parameters



(a) Q-Learning Optimal - Rewards

(b) Q-Learning Optimal

Figure 10: Q-Learning Optimal Params

4.5 25x25 Maze Environment - Modified Q-Learning:

4.5.1 Thought Process for Best Way to Solve

Strategy:

We chose to use this Q-Learning approach because we wanted a policy that *converged* towards a greedy policy. We wanted a **greedy** policy because we wanted the agent to avoid the punishment and trap zones as much as possible. We'd clearly observe the trends in these 'trap' areas after about 10 episodes which is why we chose that to guide where we placed the punishment. Tiles that got visited the most times would be where the punishment would gravitate towards in order to incentivize the agent to avoid those tiles and regions as much as possible.

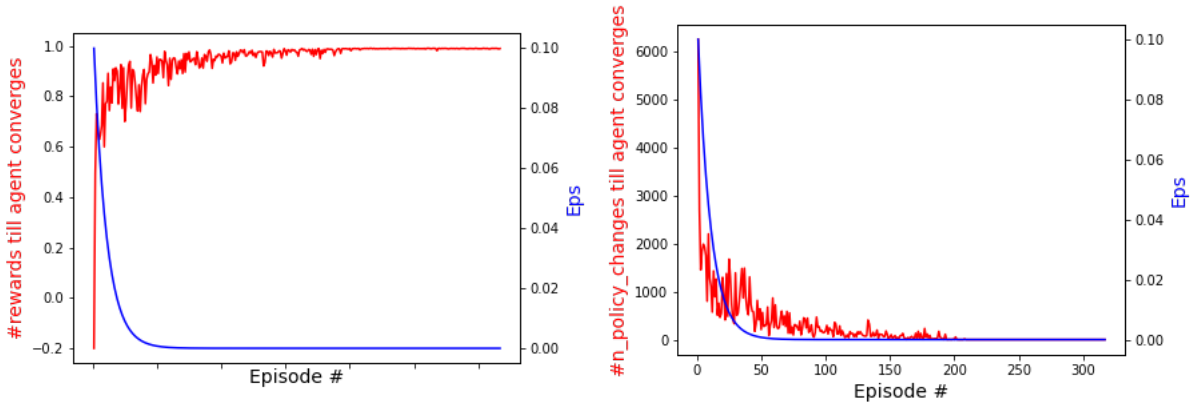
It worked at 10 episodes because we chose an exponential epsilon decay function for this part, which decayed and declined rather quickly in terms of episodes of training. We changed our epsilon decay vs. those in Exercise 2 because the board was different - 625 vs 225 - thus allowing for much more possibility to get lost and stuck. This also guided our decision to initialize with a smaller starting value for epsilon ϵ. One additional note is that the influence of the chose value for γ was also larger for this board / environment - so we chose a purposefully large value for γ in order to enable the agent to strike the balance - not risking over-exploration and being stuck in traps, but also with the higher γ, being able to 'look further ahead' to better understand where it's going, which is especially valuable and meaningful for this game since there are so many paths and so many more options for tiles, moves, combinations, and rewards.

4.5.2 25x25 Maze Summary Results

- Steps at 250, and 500 episodes $61, 57$
- Rewards at 250 and 500 episodes $0.9904, 0.9991$

4.5.3 $25x25$ Maze Discussion

When we were exploring, the algorithm got "stuck" in "hard-to-escape" areas. We wanted to help it escape these areas by marking them with "punishment". To do so, we first needed to find these bad indexes by looking at the first 10 heatmaps. Most of the routes were going through the argmax of these average heatmaps, and by marking those points we aimed to assist the agent towards the more optimal path, while also reducing wasteful exploration that risked getting stuck in a loop.



(a) $25x25$ Conv Rewards

(b) $25x25$ Conv

Figure 11: $25x25 Modified Q - Learning Convergence$

For example, we'd find an average route to see where most of the routes were going through. We observed that most of the routes didn't go through column 13, row 7. We then helped to ensure the algorithm will find other routes to send the agent through.

The GOOD reward will be at the terminal state, otherwise the Q-Learning algorithm will go over and over to it and won't converge to the terminal state. And then we were ready to start learning with the "special" rewards.

5 Ideas for Further Experimentation and Improvements

As mentioned, we could certainly evaluate a different time limit to training, have more CPU/GPU compute power available to us, and have an abundance of additional permutations of the above methodologies to further optimize the agent's strategies for various environments. There is certainly no shortage of learning here and even more room for further experimentation and improvements.

6 Code Links

Google Colab project link: mazebook_011996279_301315040

[0]Compiled with LaTeX on January 16, 2023.