

# Various Reinforcement Learning Algorithms to Solve Grid World

Matteo Esposito<sup>a</sup>, William Ngo<sup>a</sup>, Spyros Orfanos<sup>a</sup>

<sup>a</sup>*Concordia University, Montreal, Quebec*

---

## Abstract

Reinforcement Learning (RL) is widely used in different research areas to tackle problems such as resource management in computers[1], traffic light control[1], and playing games like chess[2] at a super-human level. In this project, we tackle the problem of solving a two-dimensional maze given a specific starting point. To accomplish this, three different policies were developed using RL algorithms known as SARSA, Q-Learning, and n-step SARSA. Each learning method has its theoretical tradeoffs from one another which we compare by evaluating the average training time per single run, average number of episodes until they reach optimality, and scalability by testing the algorithms on two different sized mazes. We conclude that Q-Learning is the best suited approach for Grid World since in both of our mazes, we observed consistently faster convergence rates to the optimal return and execution times.

---

## 1. Introduction

Our project, commonly referred to as “Grid World”, tackles the problem of exiting a two-dimensional maze. This problem consists of a starting point, an exit point, and a board in which the agent (player) can take actions to move up, down, left or right. This is quite a simple task for a human as one can simply inspect the entirety of the maze and, at a glance, deduce an exit strategy. However, a machine cannot readily “see” the maze or “think” of actions to take, so how can it learn to solve a maze?

To allow a machine to “understand” the problem at hand, we must first reformulate the problem into something it can understand: a sequential decision problem. A sequential decision problem consists of a set of states, a set of actions, and a reward for taking some action given the current state.



By formulating the problem in such a way, we may use Reinforcement Learning (RL) techniques. RL is learning what to do: how to map situations to actions so as to maximize some notion of cumulative reward. In other words, RL is all about discovering the best action given your current state. To do this, a numerical return is designed such that when maximized, the objective of the problem should be met. Most reinforcement learning

---

*Email addresses:* ds.mesposito@gmail.com, ID:40024121 (Matteo Esposito), ngowilliam96@gmail.com, ID:40031586 (William Ngo), spyros.orfanos1@gmail.com, ID:40032280 (Spyros Orfanos)

problems can be expressed as sequential decision problems, as they must include a set of the possible states of the environment, a set of possible actions and an objective. [3]

The goal of this project is to compare the rate of convergence of the estimated value function at the starting position,  $\hat{V}_*(S_0)$ , to the optimal policy's value function at the starting position,  $V_*(S_0)$ , of three RL methods: SARSA, Q-Learning, and n-step SARSA. This is measured in three ways: number of episodes until convergence to  $V_*(S_0)$ , run time of the algorithms, and how the algorithms scale to much larger mazes.



## 2. Problem Description

As described above, “Grid World” is a sequential decision problem as the set of states is the possible positions in the maze, the set of actions is deciding which direction to go and the objective is to find the exit of the maze in the shortest amount of steps, given a starting point and the exit point.

## 3. Analyses

### 3.1. Implementation - Code

We would like to briefly comment on the structure of our code. The project can be run entirely from `main.R`. The environment of the maze was built using object-oriented programming, which was done with the R6 package. Object-oriented programming was a core concept in the design of our environment and our code as we wanted each script to do achieve specific task. By doing this, the script that generates the environment can be used for future tasks at hand. In `main.R` all of our algorithms are called (by means of the `source()` function) and with them, average runtimes, optimal action plots and average return plots are output.

### 3.2. Environment

Here, the set of states consists of all valid board positions  $(x, y)$  where  $x \in \{1, 2, \dots, nrow\}$  and  $y \in \{1, 2, \dots, ncol\}$ . The set of actions consists of moving {Right ( $action = 1$ ), Down ( $a = 2$ ), Left ( $a = 3$ ) or Up ( $a = 4$ )} where the objective is to find the exit of the maze.

To implement our maze, we create an m-by-n matrix that stores the values 0,1 and 10. These three values each map to valid space, a wall and an exit tile, respectively. The starting point of the agent is defined as the variable: “`ratStartingPosition`” defined in the main script. This was designed in a manner in which we could easily modify the characteristics of the maze by simply modifying one or two variables. Through the use of object-oriented programming and the R6 package, we can access and modify many properties of our maze object.

Figure 1: Form of maze 1

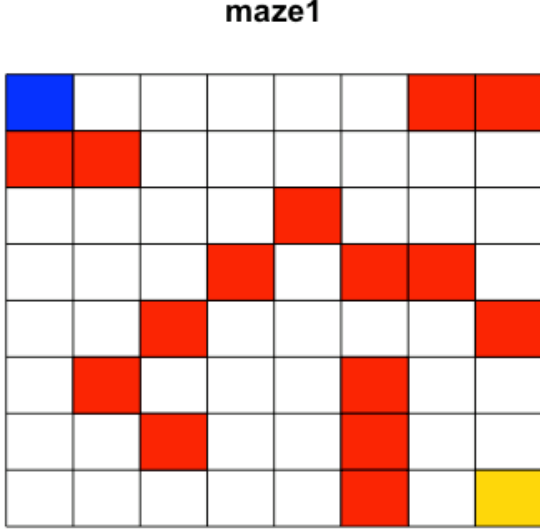
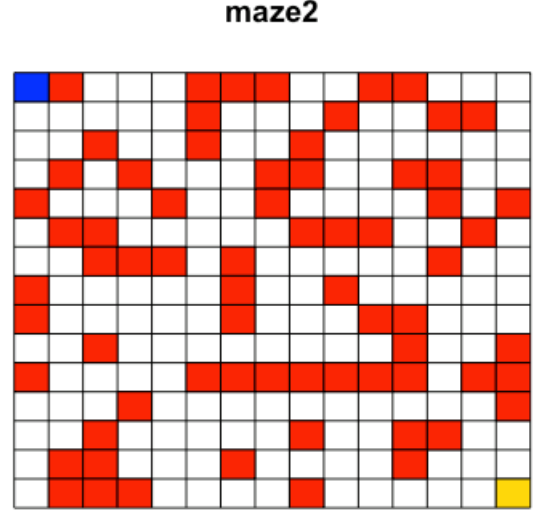


Figure 2: Form of maze 2



### 3.3. Statespace

To simplify the RL algorithms, we consider the bijective mapping  $f : \mathbb{N}^2 \rightarrow \mathbb{N}$  which maps valid board positions (x,y) to states (s). The mapping is as follows: the upper-left most corner is mapped to state 1. Then, moving right along the first row, the next valid board position is mapped to the next natural number. Once all board positions in a given row are mapped to (s), we apply the same logic to the next row, until we have mapped all valid board positions.

Note: All plots and future references represent runs that have been performed in the maze 1 environment unless stated otherwise.

For every action we assign a reward strictly dependent on the environment (set of states), with the following structure:

- Reward attributed to hitting a wall - denoted as a red tile, or going beyond the border = -0.75
- Reward attributed to visiting a new, unseen tile that isn't the exit tile = -0.05
- Reward attributed to revisiting a previously seen tile = -0.30
- Reward attributed to visiting the exit tile - denoted as a yellow tile = 5

One thing to note is that if the agent attempts to hit a wall or the outer bound of the maze then it stays at its current position and gets attributed a reward of -0.75. The design of our rewards should effectively train the agent to effectively exit the maze in the least amount of steps. Given the structure of our test maze, our optimal return would be



$G_0 = 3.85$ , which will be denoted as the orange convergence asymptote curve in our return average plots presented in the following section of this report.

### 3.3.1. Random Policy

Before running any learning algorithm, we evaluated the performance of a policy that selects actions at random. As expected, the policy performs extremely poorly as out of 10000 episodes, only 11% of the episodes were able to find the exit within less than 400 moves and for the episodes that ended, the average return is extremely poor when compared to the optimal return.

Table 1: Results of random policy after 10000 episodes

Number of episodes that ended	Average Return of the policies that ended
1075	-115.569

Using the random policy algorithm, out of 10000 episodes only 10.75% of all episodes exit the maze successfully/terminate within 400 steps, yielding an average reward of -115.596.

The above results illustrate how weak random policies are at solving grid world.

## 3.4. Algorithms

Here we describe the theoretical backgrounds and state-action value update functions of the 3 approaches we considered.

### 3.4.1. SARSA

Our second algorithm is the SARSA (State-Action-Reward-State-Action) method. The SARSA method is an on-policy temporal difference control method. It enables the agent to update the (state, value) or  $Q_\pi(s, a)$  value after every single action under a given policy  $\pi$ . Given an epsilon value, the agent will either tend to exploit the current optimal action at a given state ( $\epsilon - greedy$ ) or explore alternate actions in an effort to find different trajectories to the end goal.

Its update rule is defined as follows:

$$Q_{t+1}(S_t, A_t) \leftarrow Q_t(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)]$$

### 3.4.2. Q-Learning

Q-Learning is another temporal difference control policy and is usually used as an off-policy method. Here, we instead use it as an on-policy method by calculating an  $\epsilon$ -greedy policy based on our estimate of  $Q(S, A)$ .

$$Q_{t+1}(S_t, A_t) \leftarrow Q_t(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)]$$

### 3.4.3. *n*-Step SARSA

Finally, *n*-step SARSA which is a variation of the aforementioned SARSA algorithm as it varies the number of rewards that are estimated. It is implemented as follows:



$$\begin{aligned} Q_{t+n}(S_t, A_t) &= Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \\ Q_{t+n}(s, a) &= Q_{t+n-1}(s, a) \quad \forall (s, a) \neq (S_t, A_t) \end{aligned}$$

$$G_{t:t+n} = \sum_{j=1}^{\min(n, \tilde{T}_{I(t)}-t)} \gamma^{j-1} [R_{t+j} + \mathbb{1}_{\{t+n \leq \tilde{T}_{I(t)}\}} \gamma^n V_{t+n-1}(S_{t+n})]$$

### 3.5. *Considerations: Epsilon Selection*

Since all of our RL algorithms involve on-policy methods, the policies require exploration; however, this means the policy will converge to a policy that is  $\varepsilon$ -greedy. This presents a challenge, as we know that the optimal policy is deterministic. We explored different choices of epsilon, namely epsilons that were decreasing as the policy improved after each episode. Thus, epsilon goes to zero and hence the policy converges to the optimal deterministic policy, as desired.

To address this challenge, we tested multiple epsilon update functions on one of our learning algorithms. As illustrated in the following figure, we noticed that the update rule  $0.4/n$  didn't give the agent enough time to explore. The log update rule converged in a comparable but better rate than the constant method and our exponentially decaying rule proved to be the best suited as it promoted exploration at the beginning and then slowly never explored, exploiting the optimal action.

Figure 3: Q-Learning returns under various  $\varepsilon$ -update rules.

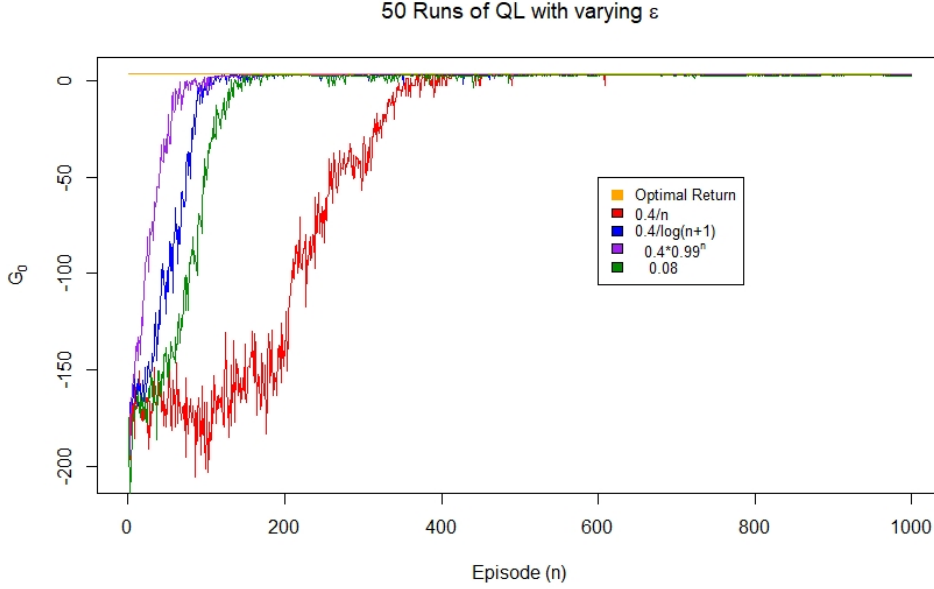


Table 2: Varying epsilon approaches runtime analysis

$\varepsilon_n$	Average # Episodes Before Convergence	Runtime (mins/ep)
$\frac{0.4}{n}$	$\sim 450$	$\sim 0.189$
$\frac{0.4}{\log(n+1)}$	$\sim 180$	$\sim 0.087$
$0.4(0.99)^n$	$\sim 120$	$\sim 0.073$
0.08	$\sim 200$	$\sim 0.109$

We observe that the average runtime per episode is proportional to the average number of the episodes before convergence which is expected since policy that was that was attributed to the update rule more quickly found the optimal policy and chose the most appropriate actions in the greedy case after converging.

Note: The reported runtime averages are hardware dependent.

### 3.6. Consideration: Bias vs. Variance Tradeoff

Another consideration of any machine **learning** project is the bias-variance tradeoff. Since we are only dealing with TD (temporal difference) methods, we will only speak about the bias-variance present in these types of reinforcement learning algorithms. Generally speaking, TD methods have lower variance but have higher bias compared to Monte-Carlo method. The lower variance is due to the fact that in the update equation for the value function of TD methods, the term/target value  $R_{t+1} + \gamma V_\pi(S_{t+1})$  shows that the state-value update

utilizes only the immediate future reward while in contrast to the Monte Carlo method, which have target value  $G_t = R_{t+1} + \gamma R_{t+2} + \dots$ , which clearly depend on a greater number of rewards.

The presence of bias in the TD methods is due to the fact that the initial value function estimate is being used to update our value function estimate. It is worth noting that even though TD methods are biased, the bias is exponentially decaying thus asymptotically it is not an issue.

N-Step SARSA is a method in which we can control the bias-variance tradeoff as the value function is updated using the next n-many rewards. The higher the n, the more rewards we utilize to update the value function and thus higher variance, vice-versa, the lower the n, the lower the variance but a higher bias is incurred.

The concept of bias-variance tradeoff is a theoretical view of these learning methods but to know which learning method is best for a given problem we have to implement them and see which performs the best. Before comparing the different learning algorithms between each other, we first vary the “N” in N-Step SARSA to determine which one seems to be performs the best. We ran and compared 5,10 and 15 step SARSA implementations on our first 8x8 maze and observed the following results:

Figure 4: Comparison of n-Step SARSA algorithms by return  $G_0$  at different n-values with exponentially decaying  $\epsilon$  values.



From the following figure we can conclude that there is no significant discrepancy between 5-step and 10-step SARSA whereas 15-step SARSA is less suited to our problem considering all 3 convergence rates. 5-step or 10-step is clearly better for this problem. We decided to use 10-step SARSA in all future analyses/plots.

### 3.7. Maze 1 Results

Figure 5: Maze 1 returns

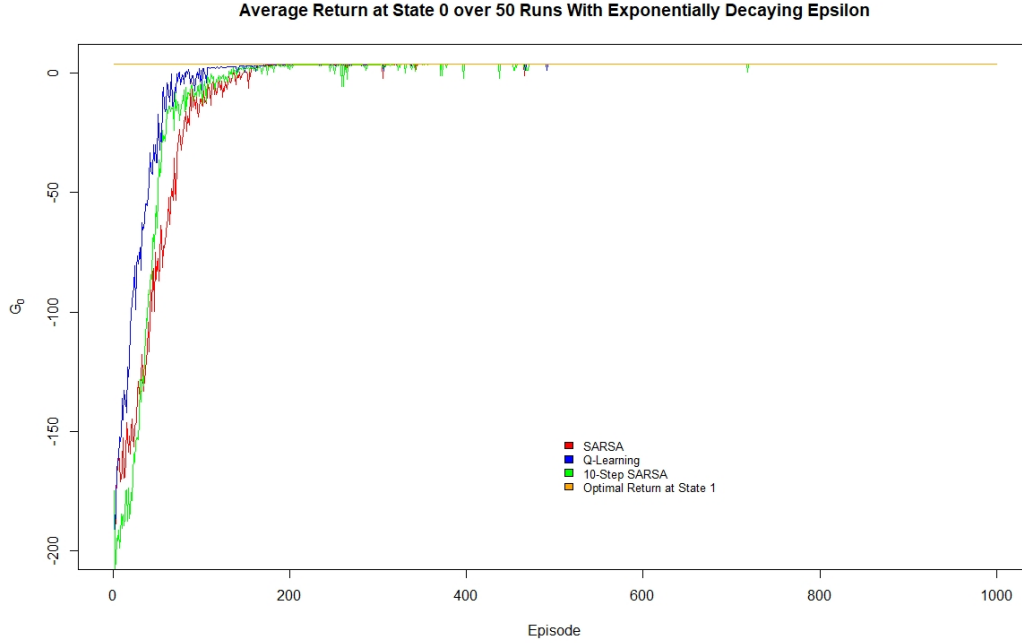


Table 3: Maze 1 runtime analysis

Algorithm	Average # Episodes Before Convergence	Runtime (mins/run)
SARSA	$\sim 200$	$\sim 0.085$
Q-Learning	$\sim 150$	$\sim 0.075$
n-Step SARSA	$\sim 200$	$\sim 0.094$

We observe that Q-Learning converges faster than SARSA and 10-step SARSA. It converges in roughly 3/4 the number of episodes and has a per episode runtime which is quicker than that of the SARSA algorithms. We can observe that the average runtime is proportional to the average number of episodes before convergence because past a certain point, when the agent selects the greedy action, it will select the optimal action and thus an episode would terminate quickly.



Figure 6: Optimal actions on maze 1 using SARSA

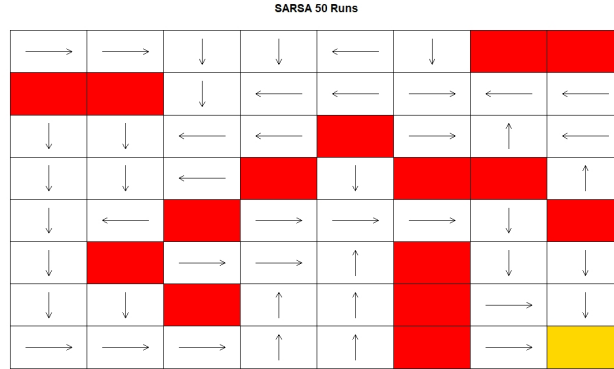


Figure 7: Optimal actions on maze 1 using Q-Learning

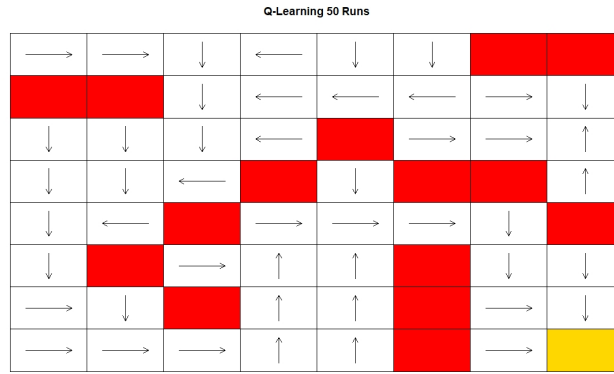
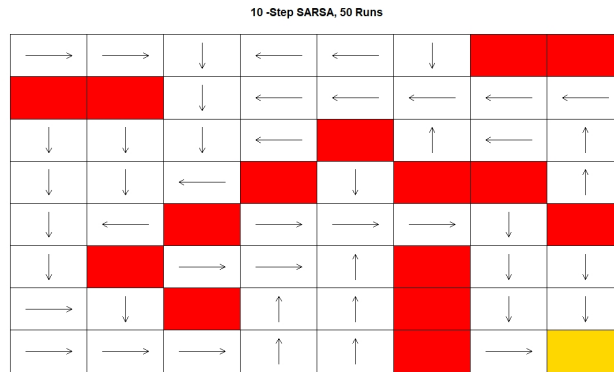


Figure 8: Optimal actions on maze 1 using SARSA



### 3.8. Maze 2 Results

Figure 9: Maze 2 returns

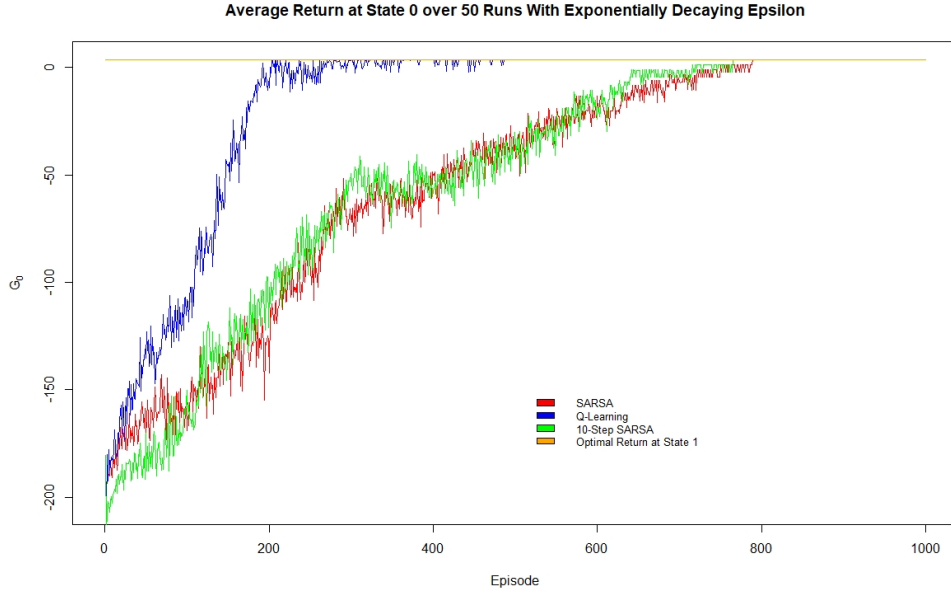


Table 4: Maze 2 runtime analysis

Algorithm	Average # Episodes Before Convergence	Runtime (mins/run)
SARSA	$\sim 800$	$\sim 0.375$
Q-Learning	$\sim 400$	$\sim 0.275$
n-Step SARSA	$\sim 800$	$\sim 0.354$

Similar to the plots in maze 1, we observe that Q-Learning converges at a much greater rate than SARSA and 10-step SARSA. It converges in roughly 1/2 the number of episodes and has a per episode runtime approximately 3/4 that of the SARSA algorithms. As expected, since we are dealing with a much larger grid relative to maze 1, there is a much larger difference in results between the algorithms.


In the maze 2 example we are dealing with a  $15 \times 15$  grid  therefore a larger discrepancy is to be expected as since there are more states and episodes last longer, the different learning algorithms have a larger room to distinguish themselves from one another.

Figure 10: Optimal actions on maze 2 using Q-Learning



#### 4. Conclusion

Q-Learning (QL) was the most effective policy based on 3 factors, scalability, average runtime and average return per policy (or speed to convergence to optimal return per maze). We found that in maze 1, QL converged in  $\sim 150$  episodes compared to the SARSA which converged in  $\sim 200$  episodes with runtimes with an average runtime 0.015min/run slower than QL. **on** maze 2, roughly twice the size of maze 1, the difference became much clearer. QL converged in  $\sim 400$  episodes compared to the SARSA which converged in  $\sim 800$  episodes with runtimes with an average runtime 0.09min/run slower than QL.

A future avenue of research that could benefit our approach would be addressing the issue of not commonly visited states having sub-optimal actions as final optimal actions. This could be solved with the use of deep learning approaches such as neural networks. As in the case of our  $8 \times 8$  maze, we see that once the agent found the optimal path, and ceased to explore at the same rate as prior to finding the best route, whatever it had set as the optimal action in some of the states actually isn't optimal. Another interesting topic to explore is the use of a decreasing  $\varepsilon$  in the "Grid World with Cliff" problem. This is particularly interesting the  $\varepsilon$ -greedy policy often takes a "safer" route to avoid falling off the cliff when taking an exploratory action. However, when using a decreasing  $\varepsilon$ , should we expect the policy to converge to the true optimal policy, or has agent learned to avoid this path?

## 5. Appendix A: Choice of Rewards

In an attempt to accelerate the convergence to  $V_*(S_0)$ , we decided to penalize revisiting an already seen state more severely than visiting an unseen state. The idea is that the agent has more incentive to explore different paths instead of wasting time visiting states it has already seen. We realize this could lead to the agent taking “detours” (i.e., suboptimal actions) to avoid the costly reward of visiting an already seen state. This could explain the suboptimal action choices for states rarely visited (e.g.: in the upper-right part of maze1). However, for states visited sufficiently often, this choice of rewards does not present a problem as demonstrated by the convergence of our algorithms.

Below we present the results of the algorithms (with an exponentially decaying epsilon) when we set both the reward of visiting an unseen tile and a seen tile to -0.05. We see that the agent finds the optimal action for almost all states; however, convergence to  $V_*(S_0)$  is slower. Since we are only interested in the latter, the choice of rewards is further justified.



Figure 11: SARSA with  $r = -0.05$



Figure 12: Q-Learning with  $r = -0.05$

Figure 13: 10-step SARSA with  $r = -0.05$

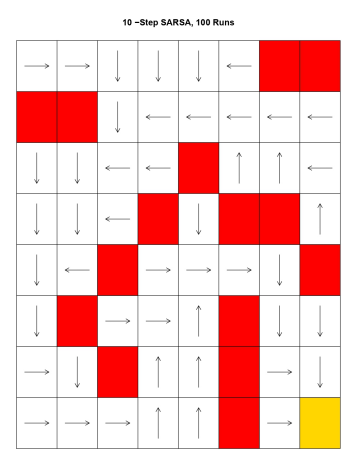
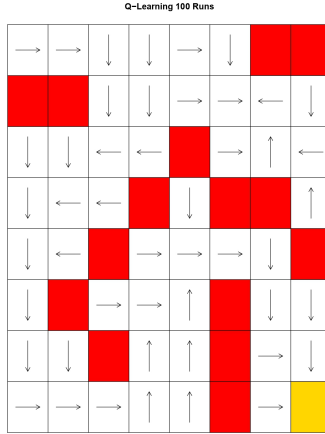
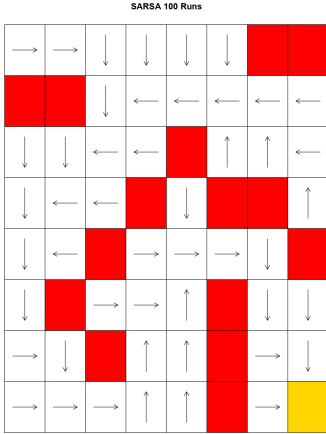
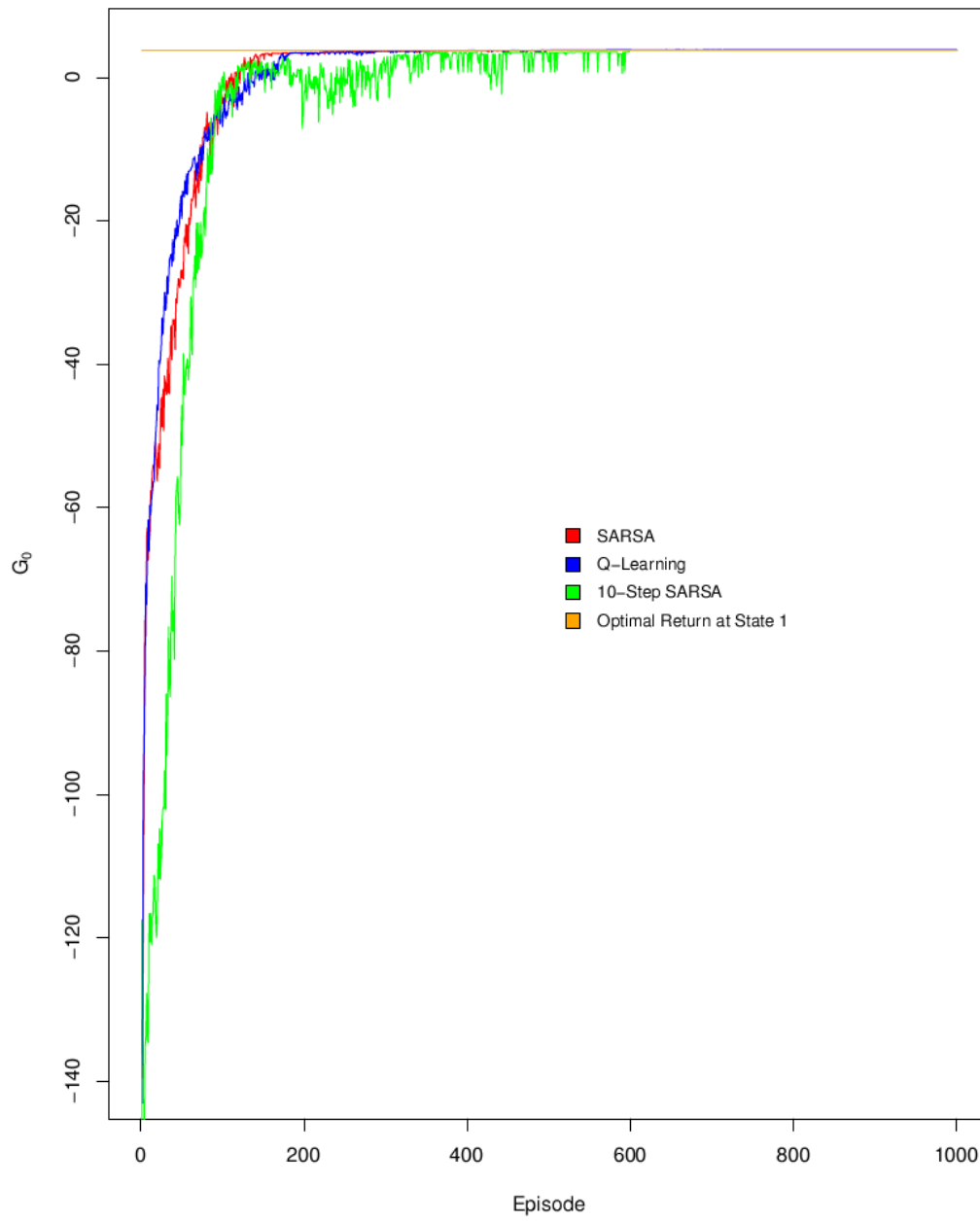


Figure 14: Average return at state 1 over 50 runs with exponentially decaying epsilon.



## 6. Appendix B: Additional Maze 2 Results

Figure 15: Optimal actions on maze 2 using SARSA

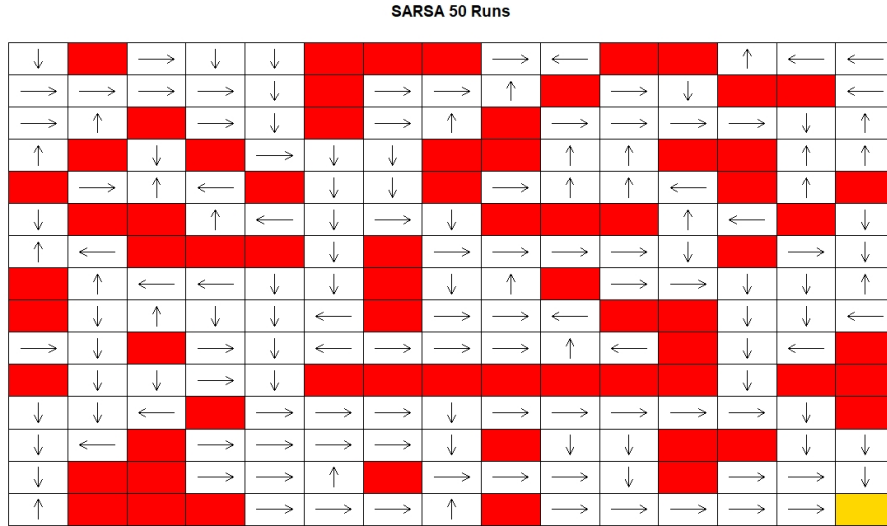
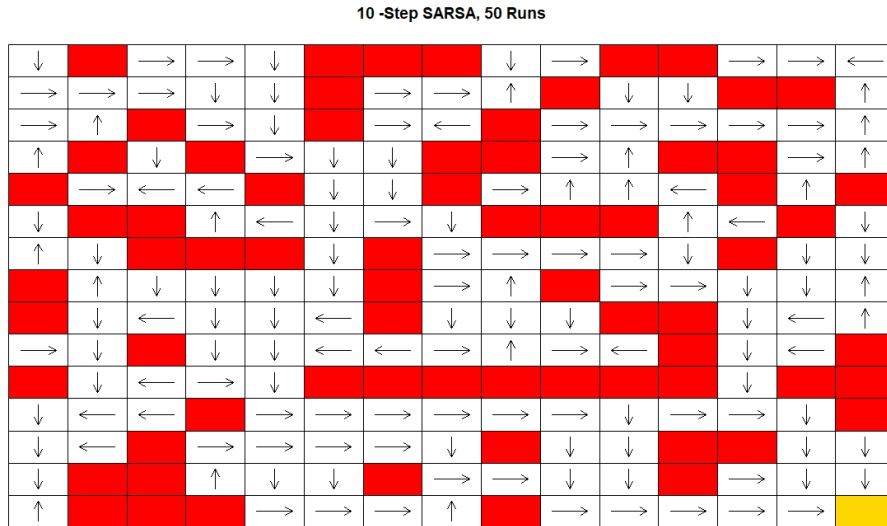


Figure 16: Optimal actions on maze 2 using 10-step SARSA



## References

- [1] Applications of reinforcement learning in real world, Towards Data Science (Aug 2018).  
URL <https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12>
- [2] Alphago zero: Learning from scratch, DeepMind.  
URL <https://deepmind.com/blog/alphago-zero-learning-scratch/>
- [3] F. Godin, Stat497 reinforcement learning slides chapter 1, Concordia University (2019).