FINAL HOMEWORK ASSIGNMENT

MULTI AGENT SYSTEMS

MSC ARTIFICIAL INTELLIGENCE

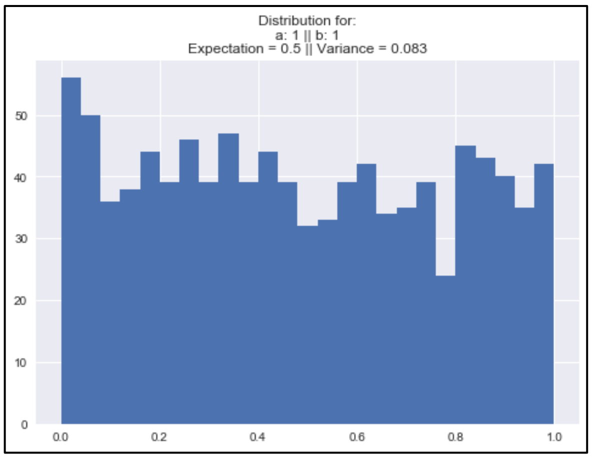
VRIJE UNIVERSITEIT

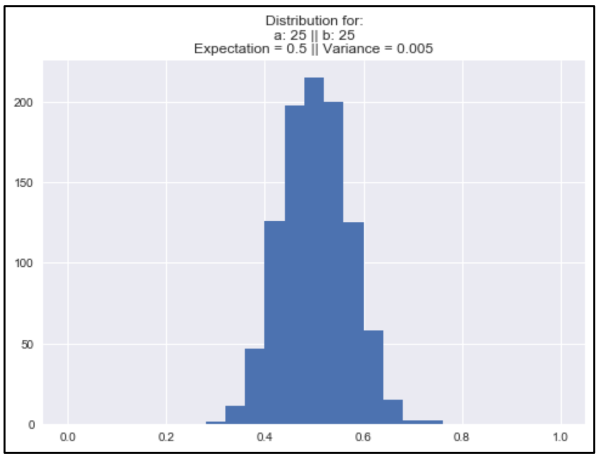
MICK IJZER – 2552611

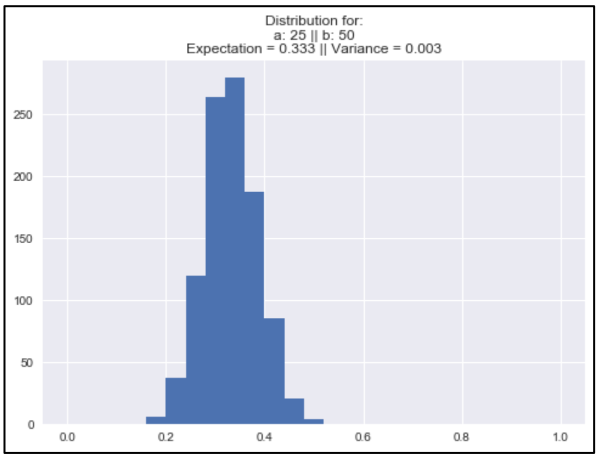
1 MULTI-ARMED BANDITS

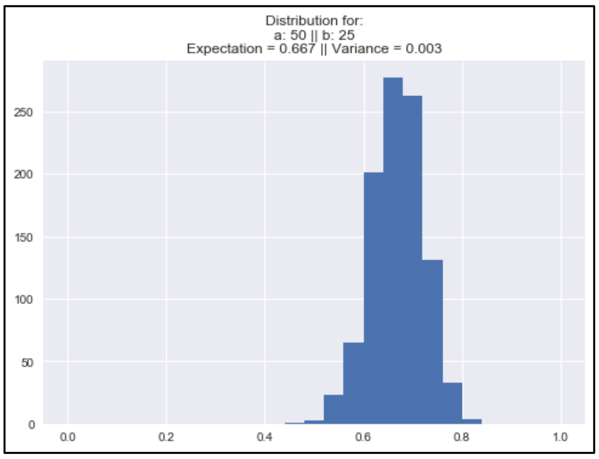
1.1 PROPERTIES OF THE BETA-DENSITY

The properties of the beta density are shown in the four images below. When the parameters **a** and **b** are both 1 the density looks like the uniform distribution (top left graph). When both **a** and **b** are increased but remain equal the expectation remains the same, however the variance decreases and peak around the expectation becomes visible (top right graph). When **a** and **b** are not equal the distribution becomes skewed, to the right if **b** is larger than **a** (bottom left graph) and to the left if **a** is larger than **b** (bottom right graph). In the graphs below the **a**, **b**, expectation, variance and the result of 1.000 samples are shown.



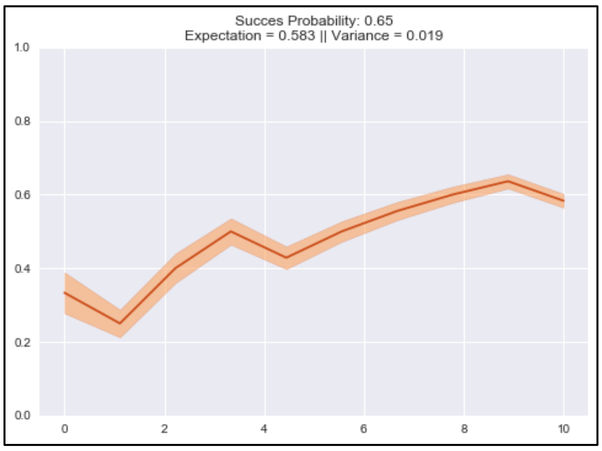


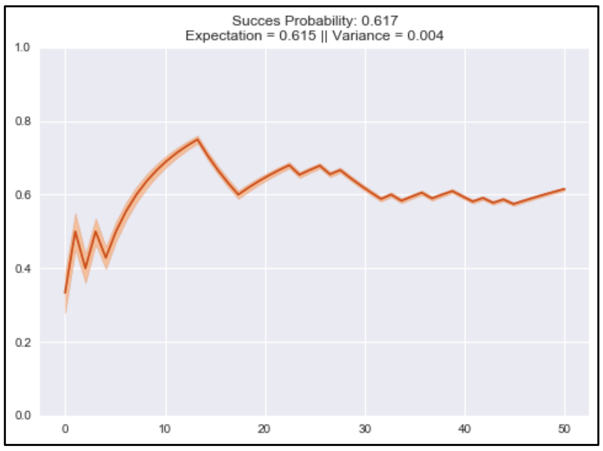


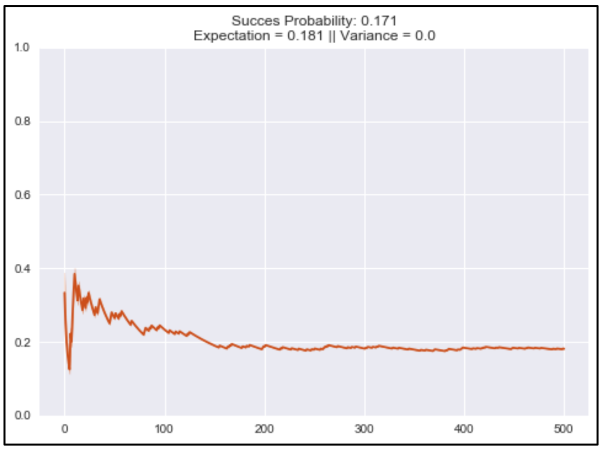


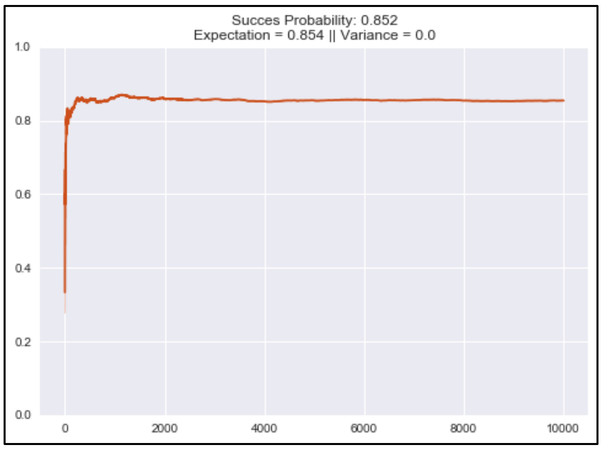
1.2 THOMPSON UPDATE RULE

In the four graphs below the results of using the Thompson update rule to approximate the success probability is shown. In all of the graphs the expectation is plotted, and the variance is represented by the colored area around the line. The expectation and variance are computed by using the formulae of the expectation and variance based on the actual values of **a** and **b**. The graphs differ in the number of samples, because of this in both graphs at the top the variance can be seen clearly. In the two graphs at the bottom the number of samples is high enough that the variance approximates 0 and can thus not be seen clearly in the graph. As can be seen in the graphs the expectation of the success probability fluctuates for the first ~200 samples, after which the expectation stabilizes around the true success probability.



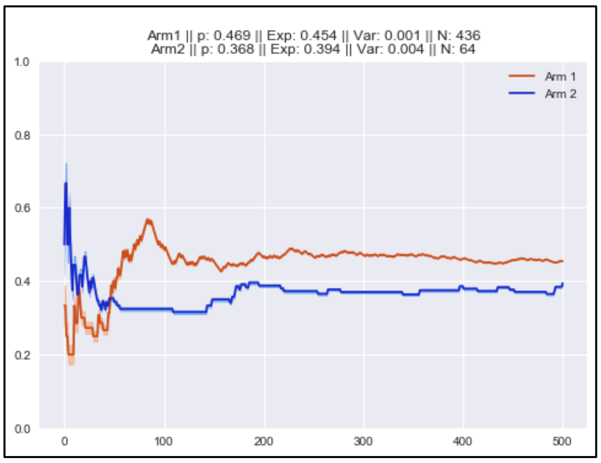


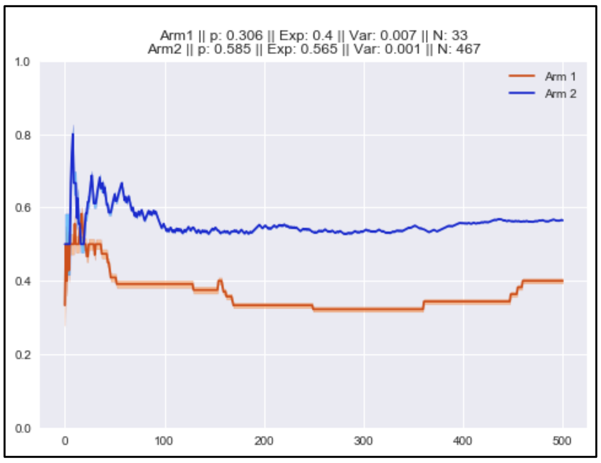


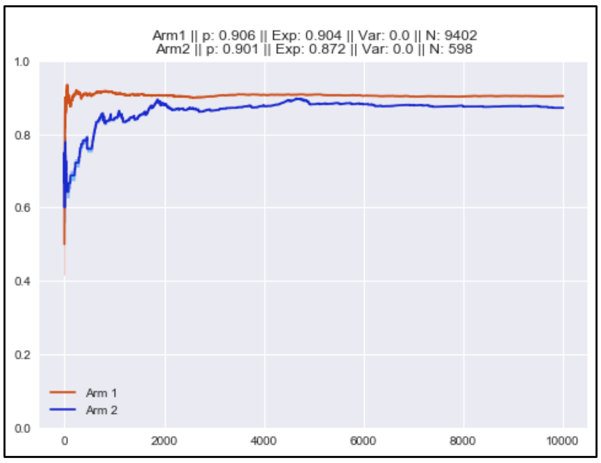


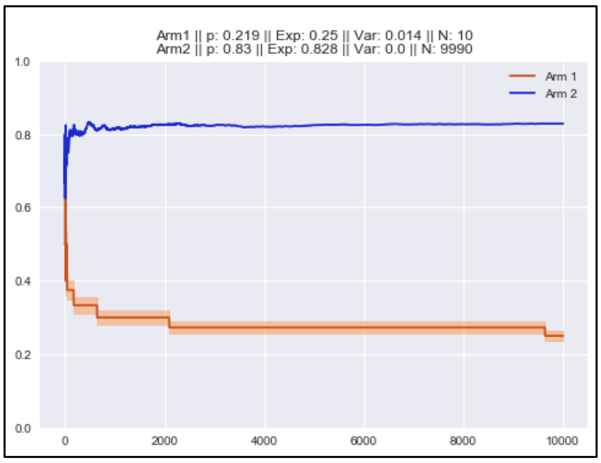
1.3 THOMPSON SAMPLING

By implementing Thompson Sampling for the scenario with 2 bandits the graphs below were created. This method required initialization so that $a_1 = b_1 = 1 = a_2 = b_2$. First a sample was drawn from the beta distributions of both bandits, after which the bandit is chosen whose sample was highest. The a and b from the chosen bandit are updated according to the reward. This means that even though the expectation of the first bandit may be a lot higher it is still possible that the sample from the first bandit is lower than the second bandit and that the sample is therefore drawn from the 'non-optimal' bandit. The graphs below show the fluctuations in expectation of the bandits over time. The top two graphs show 500 iterations and the bottom two show 10.000 iterations. The 'flat' areas represent the time that the other bandit is continuously sampled, which means that the expectation from the 'flat' bandit doesn't get updated during that time.









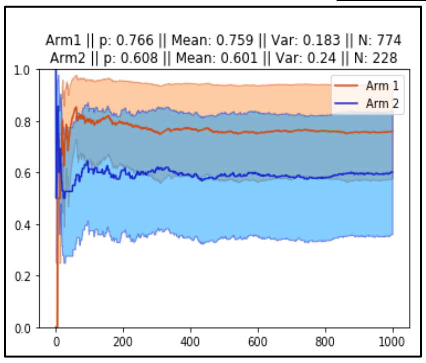
1.4 UCB & E-GREEDY

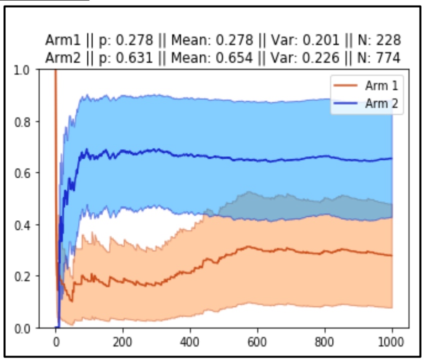
To compare the performance of the Upper Confidence Bound (UCB), Epsilon-Greedy (e-greedy), and Thompson Sampling (Thompson) an experiment using these methods was carried out. For the e-greedy method epsilon was set to 0.1 so that there is a 10% chance to choose the non-optimal arm. Furthermore, 6 different probability combinations for the two arms were used to see if the difference in performance of the methods depends on the probability combinations. The 6 combinations were (0.2/0.8), (0.3/0.7), (0.4/0.6), (0.5/0.5), (0.2/0.4), and (0.7/0.9) for the first and second bandit respectively. A few examples of the UCB and e-greedy methods are shown in the graphs at the bottom of the page. To increase the power of these comparisons every method was used 25 times for every probability combination. The metric that was used to measure performance was the mean accumulated reward after 1.000 iterations.

In the table below the results of this experiment are schematically shown. The columns indicate the method that was used and the rows the different probability combinations. For example, the mean accumulated reward for the Thompson method in the 0.2/0.8 scenario was 798.08 (STD = 12.17). In this scenario the expected accumulated reward for a perfect method is 800 (1.000 * 0.8), so the Beta method performs quite well. Overall Thompson Sampling seems to perform the best, followed by UCB and e-greedy seems to perform the worst.

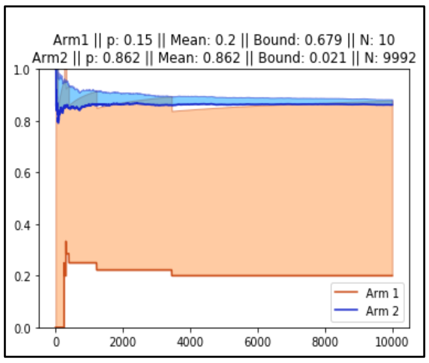
	Thompson		UCB		e-greedy	
	M	STD	M	STD	M	STD
0.2/0.8	798.08	12.17	792.32	16.10	736.92	12.52
0.3/0.7	698.72	12.88	694.24	15.72	662.64	15.78
0.4/0.6	595.64	13.30	593.28	15.32	577.96	19.21
0.5/0.5	501.24	13.43	508.52	17.04	506.68	17.39
0.2/0.4	394.84	18.42	393.00	18.86	377.08	14.15
0.7/0.9	899.08	12.03	890.00	9.15	881.96	12.3

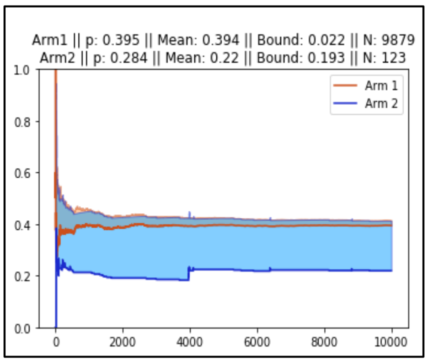
E - Greedy





UCB





2 REINFORCEMENT LEARNING: CLIFF WALKING

2.1 SARSA & Q-LEARNING

To compare SARSA and Q-Learning an epsilon-greedy policy was used. The values for the relevant parameters and experimental settings are as follows:

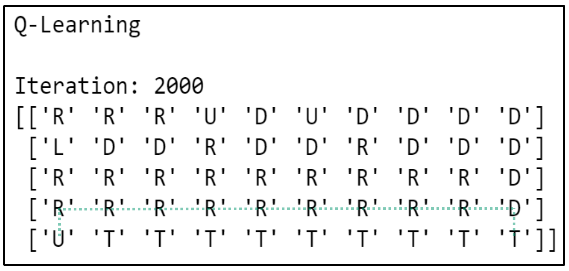
- Epsilon = 0.05
- Learning rate = 0.5
- Discounting rate = 0.9
- Iterations = 2.000

The regular update function for SARSA and Q-Learning were used, where alpha is the learning rate and gamma is the discounting rate:

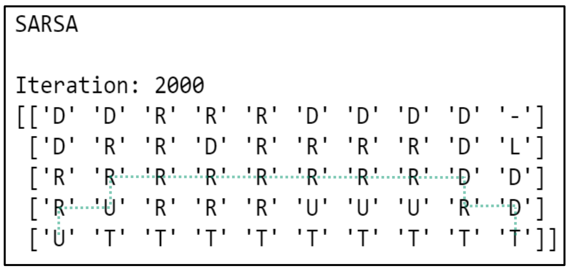
- Q-Learning: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
- SARSA:’ $Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma Q(S', A') - Q(S, A)\right)$

In the diagrams below the policy that is the result of 2.000 runs is shown for Q-Learning and SARSA. The green dotted line shows the path one would take if one would follow the policy. As can be seen the optimal path from Q-Learning goes directly along the cliff, while the optimal SARSA strategy moves one row above the cliff and is therefore ‘safer’. This is because in Q-Learning the Q(s_{t+1}, a) is the optimal action, this means that the Q(s_{t+1}, a) that is used in the update is never the reward from the cliff. However, this is the case in the Q(s_{t+1}, a) from SARSA, because of the epsilon-greedy strategy it is possible for a value to be updated where Q(s_{t+1}, a) is the reward of the cliff (-100). This explains why the optimal policy for SARSA is ‘safer’, since the reward from the cliff is carried over to adjacent states.

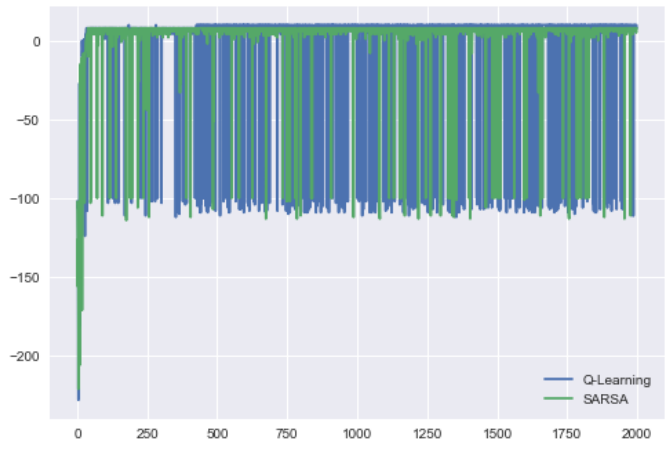
The consequence of these differences between SARSA and Q-Learning is that Q-Learning has a higher performance in situations where the strategy is greedy and the state-action has a deterministic result. In the graph at the bottom of the page the rewards for both methods over time are shown. In this graph the difference between the methods can be seen; with Q-Learning the amount of times the cliff is encountered is higher in the later iterations due to the ‘risky’ cliff walking. On the other hand when Q-Learning does reach the terminal state the reward is higher, because of the shorter path that is taken.



```
Q-Learning

Iteration: 2000
[['R' 'R' 'R' 'U' 'D' 'U' 'D' 'D' 'D' 'D']
 ['L' 'D' 'D' 'R' 'D' 'D' 'R' 'D' 'D' 'D']
 ['R' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'D']
 ['R' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'D']
 ['U' 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T']]
```



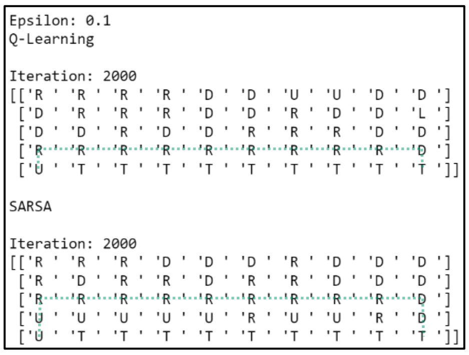
```
SARSA

Iteration: 2000
[['D' 'D' 'R' 'R' 'R' 'D' 'D' 'D' 'D' '-']
 ['D' 'R' 'R' 'D' 'R' 'R' 'R' 'R' 'D' 'L']
 ['R' 'R' 'R' 'R' 'R' 'R' 'R' 'R' 'D' 'D']
 ['R' 'U' 'R' 'R' 'R' 'U' 'U' 'U' 'R' 'D']
 ['U' 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T']]
```



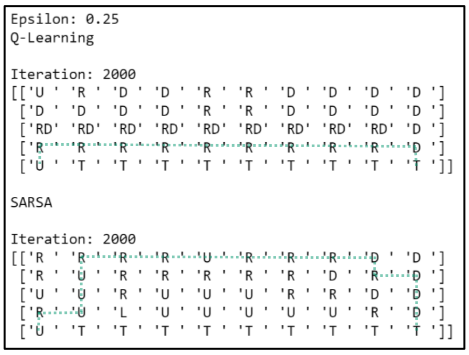
2.2 EPSILON IN E-GREEDY

For this question the values for parameters and experimental settings remain the same as the previous question, except for the epsilon for the e-greedy strategy. The effect of the epsilon is tested by repeating the previous experiment with 0.10, 0.25, and 0.50 as values for epsilon. The policies are shown in the image below, every block is a different value for epsilon and contains the policy for Q-Learning and SARSA after 2.000 runs.

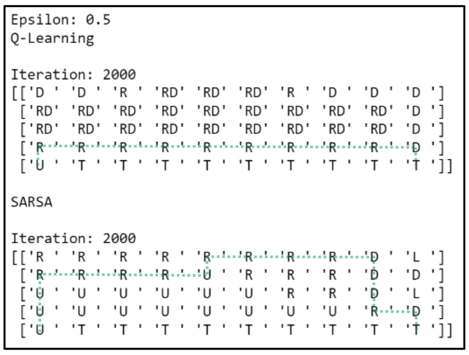
The image shows that the policy following Q-Learning remains the same independent of epsilon. The top area of the grid does slightly change, but that is the result of to the amount of data that is collected due to the higher rate of exploration. The policy remains constant because, there is no effect the cliff in the update rule (see explanation in the previous question). However, the SARSA policy does change when epsilon increases, since it moves further away from the cliff when epsilon is higher. This shift is caused by the further carry-over effect of the cliff to adjacent states (see explanation in the previous question). Compared to smaller values for epsilon the drift away from the cliff is bigger for larger values of epsilon, because the probability of Q(s_{t+1}, a) being the reward of the cliff becomes larger.



```
Epsilon: 0.1
Q-Learning

Iteration: 2000
[['R ' 'R ' 'R ' 'R ' 'D ' 'D ' 'U ' 'U ' 'D ' 'D ']
 ['D ' 'R ' 'R ' 'R ' 'D ' 'D ' 'R ' 'D ' 'D ' 'L ']
 ['D ' 'D ' 'R ' 'D ' 'D ' 'R ' 'R ' 'R ' 'D ' 'D ']
 ['R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'D ']
 ['U ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ']]

SARSA

Iteration: 2000
[['R ' 'R ' 'R ' 'D ' 'D ' 'D ' 'R ' 'D ' 'D ' 'D ']
 ['R ' 'D ' 'R ' 'R ' 'D ' 'R ' 'R ' 'D ' 'D ' 'D ']
 ['R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'D ']
 ['U ' 'U ' 'U ' 'U ' 'U ' 'R ' 'U ' 'U ' 'R ' 'D ']
 ['U ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ']]
```



```
Epsilon: 0.25
Q-Learning

Iteration: 2000
[['U ' 'R ' 'D ' 'D ' 'R ' 'R ' 'D ' 'D ' 'D ' 'D ']
 ['D ' 'D ' 'D ' 'D ' 'R ' 'R ' 'D ' 'D ' 'D ' 'D ']
 ['RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'D ']
 ['R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'D ']
 ['U ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ']]

SARSA

Iteration: 2000
[['R ' 'R ' 'R ' 'R ' 'U ' 'R ' 'R ' 'R ' 'D ' 'D ']
 ['R ' 'U ' 'R ' 'R ' 'R ' 'R ' 'R ' 'D ' 'R ' 'D ']
 ['U ' 'U ' 'R ' 'U ' 'U ' 'U ' 'R ' 'R ' 'D ' 'D ']
 ['R ' 'U ' 'L ' 'U ' 'U ' 'U ' 'U ' 'U ' 'R ' 'D ']
 ['U ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ']]
```



```
Epsilon: 0.5
Q-Learning

Iteration: 2000
[['D ' 'D ' 'R ' 'RD' 'RD' 'RD' 'R ' 'D ' 'D ' 'D ']
 ['RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'D ']
 ['RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'RD' 'D ']
 ['R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'D ']
 ['U ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ']]

SARSA

Iteration: 2000
[['R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'R ' 'D ' 'L ']
 ['R ' 'R ' 'R ' 'R ' 'U ' 'R ' 'R ' 'R ' 'D ' 'D ']
 ['U ' 'U ' 'U ' 'U ' 'U ' 'U ' 'R ' 'R ' 'D ' 'L ']
 ['U ' 'U ' 'U ' 'U ' 'U ' 'U ' 'U ' 'U ' 'R ' 'D ']
 ['U ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ' 'T ']]
```