Intelligent Agents in Machine Learning

COSC 4368 Artificial Intelligence 04/26/2022

Matthew Teal, Anil Shanker, Noah Gori, Rodrigo Mariaca

Objective

Given a 5x5, two dimensional world, two agents must pick up and drop off blocks in the corresponding cells until all blocks are completed.

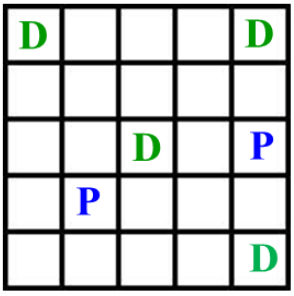

Figure 1.1 Visualization of the Pick up Drop off World

Constraints

- Agents can only move north, south, east, west. Aside from these, they may also pick up or drop off blocks.
- An agent cannot occupy the cell of another agent.

Statespace

(i, j, x, m, s, t, u, v)

(i, j, x) = (xOfAgent, yOfAgent, isCarrying)

m = directionIfAdjacentToOtherAgent

(s, t, u, v) = boolean based on x and if position is accepting/giving

Policies

PRandom - Determines the move of an agent based on random selection.

PExploit - Chooses the move with the highest Q-value 80% of the time. The remaining 20% of the time, choose the next move based on random selection. Ties are broken by random selection between the same values.

PGreedy - Chooses the move with the highest Q-value always. Ties are broken by random selection between the same values.

Experiment 1 *Matthew Teal*

Analyzing the impact of policies using the Q-learning algorithm.

Procedure

The agents should make 8,000 moves with a learning rate of 0.3 and a discount rate of 0.5. For all three runs, PRandom will use the first 500 moves. In the first run, PRandom will be used for the remaining 7,500 moves. In the second run, PExploit will be used for the remaining 7,500 moves. In the third run, PGreedy will be used for the remaining 7,500 moves.

Interpretation

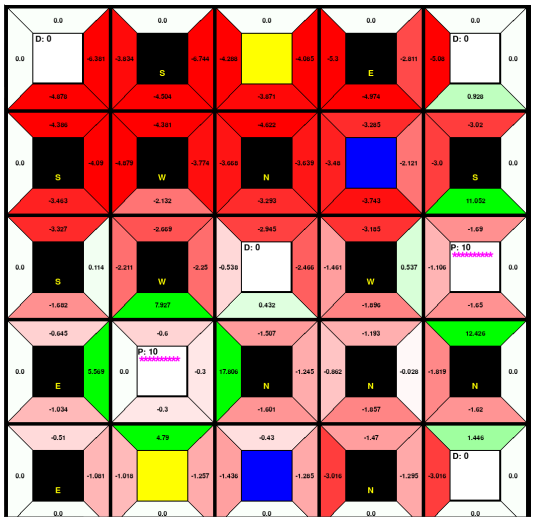

Figure 2.1 Final male Q-Values using the PGreedy Policy for the first run

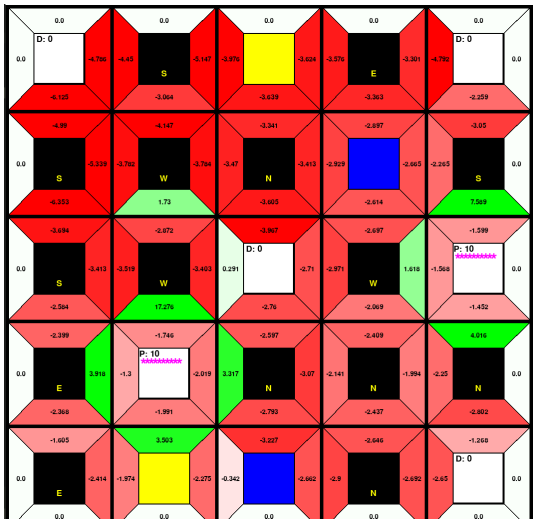

Figure 2.2 Final female Q-Values using the PGreedy Policy for the first run

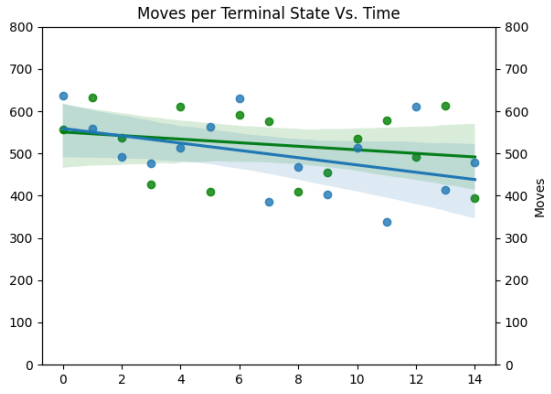

Figure 2.3 Moves per terminal state for the PRandom Policy

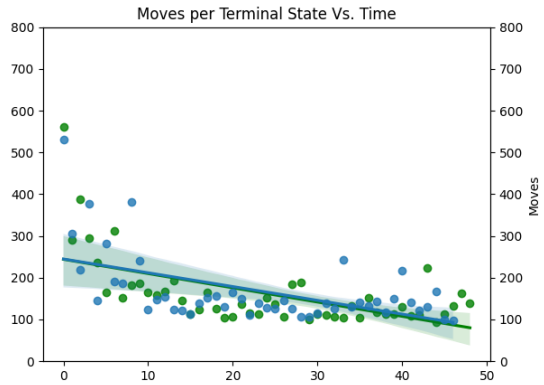

Figure 2.4 Moves per terminal state for the PExploit Policy

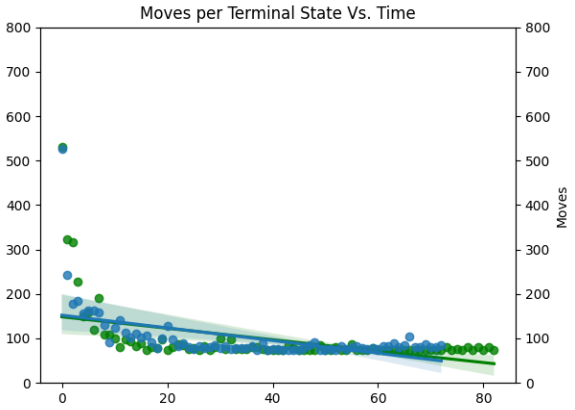

Figure 2.5 Moves per terminal state for the PGreedy Policy

	Number of Terminal States		Average Number of Moves		Minimum Number of Moves		Average Manhattan Distance	
	1st Run	2nd Run	1st Run	2nd Run	1st Run	2nd Run	1st Run	2nd Run
PRandom	15	15	521.2	498.67	394	338	1.69	1.68
PExploit	49	47	162.12	168.94	94	98	1.71	1.64
PGreedy	83	73	95.88	100.9	74	74	1.87	1.74

Figure 2.6 Number of terminal states, average moves and minimum moves for each policy run

PRandom policy is very unpredictable in performance as its title suggests. Figure 2.3 shows how sporadic the performance is within a terminal state. It has no indication whether it is going to better or worsen its efficiency. This is great for exploring the world, but not for identifying solutions. PRandom as the first 500 moves is important to the algorithm as a whole because it allows the agents to develop a sense of their surroundings. The 8,000 moves serves as a foundation to understand the effectiveness of the PExploit and PGreedy policies.

PExploit finds efficient solutions while still allowing for the chance for exploration. This is seen in Figure 2.4 in that it finds optimal paths very quickly, but periodically takes more steps than the average in a terminal state. It occurs due to the policy having a 20% chance to not choose the best applicable move.

The best performance given the learning rate and discount using a Q-Learning algorithm belongs to the PGreedy policy. It achieved over 60% more terminal states than PExploit, averaged lowest in number of moves per terminal state by a significantly large margin, and had the least number of moves per terminal state. In Figure 2.5, there is a clear trend in the agents being able to better their efficiency as they progress. The PRandom plot shows that there is no common trend between the runs meaning it could worsen, better, or remain relatively stagnant in performance.

The agents coordinate very well in the PExploit policy, but even better in the PGreedy policy. Each agent does not interfere with each other at all within the PGreedy policy. As the most efficient path is learned, it continuously follows that solution while occasionally deviating due to the Q-values changing. It should be noted that PGreedy did have a bottleneck. As it found the most fruitful paths, it would get into situations where the reward equation would return the same value it started with. This resulted in a loop of the agents' movements that they could not escape. These results reveal that PGreedy is effective in finding optimal solutions quickly, but it can be limited by its lack of exploration.

Experiment 2 *Rodrigo Mariaca*

Comparing the performance between Q-learning and SARSA

Procedure

The agent should make 8,000 moves using the SARSA algorithm. The first 500 moves should use the PRandom policy while the other 7,500 should use the PExploit Policy.

The SARSA algorithm is a variation of the Q-Learning algorithm that functions identically, except it considers the next selected action (and not solely the current action) in computing utilities.

Results:

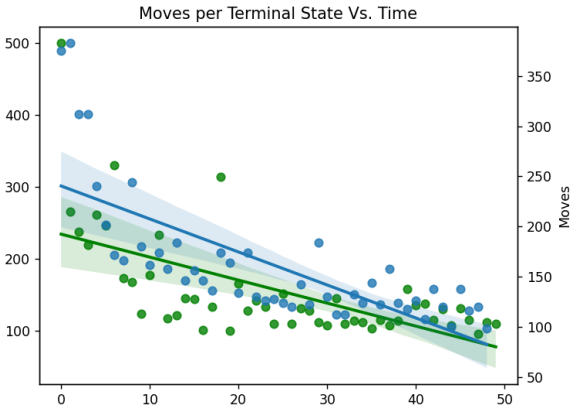

Figure 3.1 Moves per terminal state for separate Q-Tables

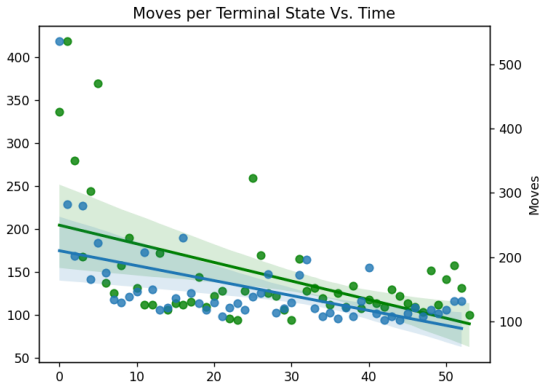

Figure 3.2 Moves per terminal state for same Q-Table

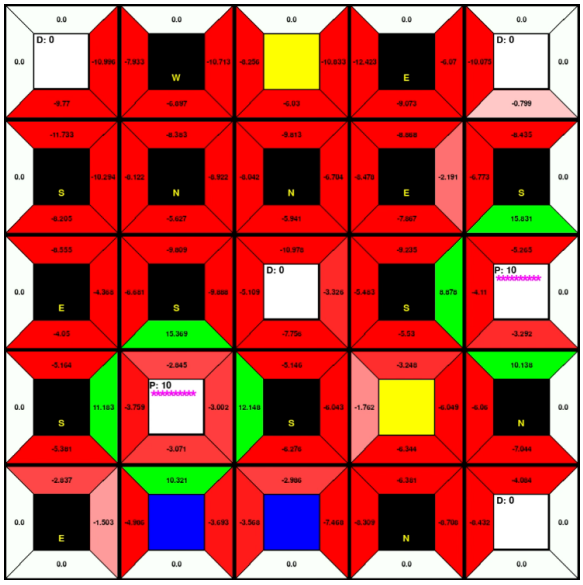

Figure 3.3 Final Q-Values using female Qtables and X=0

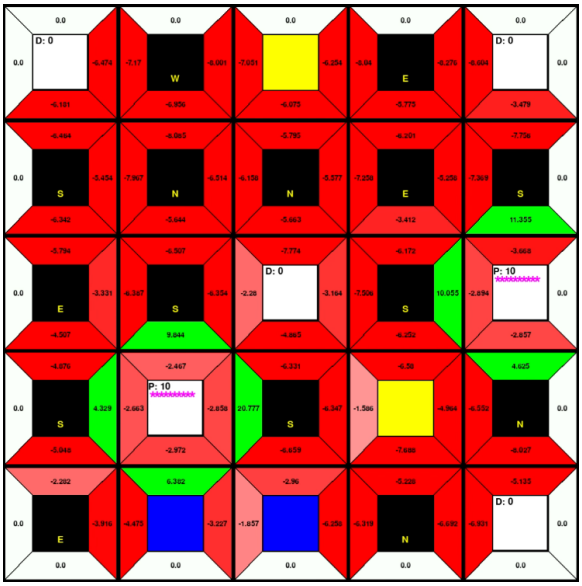

Figure 3.4 Final Q-Values using male Qtables and X

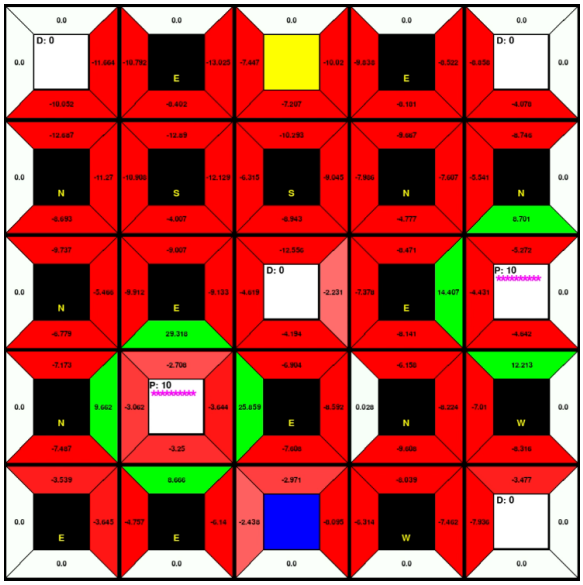

Figure 3.5 Final Q-Values using combined Qtables and X=0

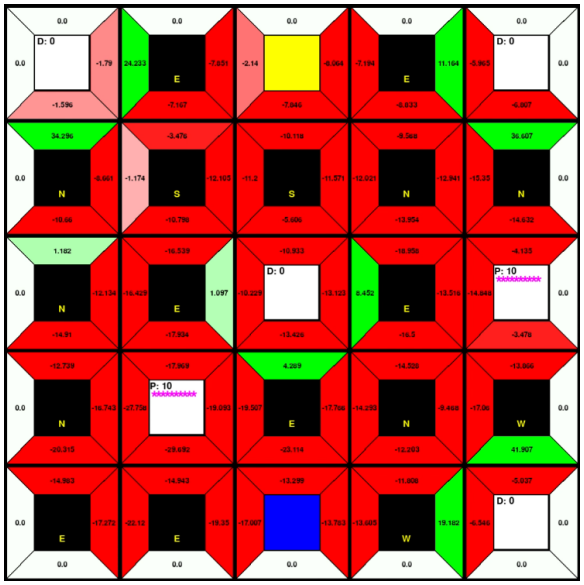

Figure 3.6 Final Q-Values using combined Qtables and X=1

Algorithm	Number of Terminal States		Average Number of Moves		Minimum Number of Moves	
	1st Run	2nd Run	1st Run	2nd Run	1st Run	2nd Run
SARSA	50	49	156.36	161.43	96	98
Q-Learning	49	47	162.12	168.94	94	98

Figure 3.7 Number of terminal states, average moves and minimum moves for each SARSA and Q-Learning using PExploit and separate Q-Tables

Algorithm	Number of Terminal States		Average Number of Moves		Minimum Number of Moves	
	1st Run	2nd Run	1st Run	2nd Run	1st Run	2nd Run
SARSA	54	53	147.41	149.62	94	102
Q-Learning	57	55	139.40	145.27	98	98

Figure 3.8 Number of terminal states, average moves and minimum moves for each SARSA and Q-Learning using PExploit and combined Q-Tables

Average Manhattan	Separate - Run1	Separate - Run2	Combined - Run1	Combined - Run2
	3.10925	3.174	3.070875	3.009

Figure 3.9 Average Manhattan Values across runs with SARSA

Figure 3.5 shows that SARSA has slightly better results compared to Q-Learning. SARSA's highest number of terminal states was 50 while Q-Learning's was 49. SARSA had a lower average number of moves with 156.36 while Q-Learning's lowest was 162.12. However, SARSA had a higher minimum number of moves at 96 while Q-Learning had 94.

As a result, the data clearly indicates that SARSA was a better algorithm. The lower average number of moves while having higher minimum number of moves

indicates that the agents were able to learn the optimal path much faster than with Q-Learning. This can also be seen when comparing the regression line from figure 3.1 and figure 3.4. As a result, the data clearly indicate that SARSA is a superior learning algorithm in this particular agent coordination state space. This is likely due because SARSA is an algorithm that considers both the immediate action chosen as well as the action from the next state. SARSA performs more favorably because this state space involves agent coordination.

Q-learning only considers the optimal choice of action based on the current state space, causing each agent to make more “selfish” choices by reducing the effect of the future state’s value. On the other hand, SARSA is an on-policy algorithm that also considers the next state’s action. Consequently, SARSA seems to perform better in the state space because it allows for the agents to better coordinate with each other. Based on the experimental data, we can plausibly infer that this leads to more efficient pathfinding, allowing us to reach more terminal states within the same number of moves.

Experiment 3 *Anil Shanker*

Analyzing the impact of the learning rate on the Q-learning algorithm.

Procedure

Each agent should make 8000 total moves. The first 500 steps are made using the PRandom policy. Afterward, the agents switch to the PExploit policy for the remainder of the algorithm. The experiment will be run twice, with different random seeds to ensure true randomness. Furthermore, the first run will use a learning rate of 0.15, and the second will use a learning rate of 0.45. The algorithm used was SARSA.

Results:

Learning Rate	Number of Terminal States		Average Number of Moves		Minimum Number of Moves	
	1st Run	2nd Run	1st Run	2nd Run	1st Run	2nd Run

0.3	50	49	156.36	161.43	96	98
0.15	44	43	180.68	185.77	108	110
0.45	50	48	158.96	165.04	96	96

Figure 4.1 Number of terminal states, average moves, and minimum moves for SARSA using PExploit and Learning Rate of 0.3, 0.15, and 0.45.

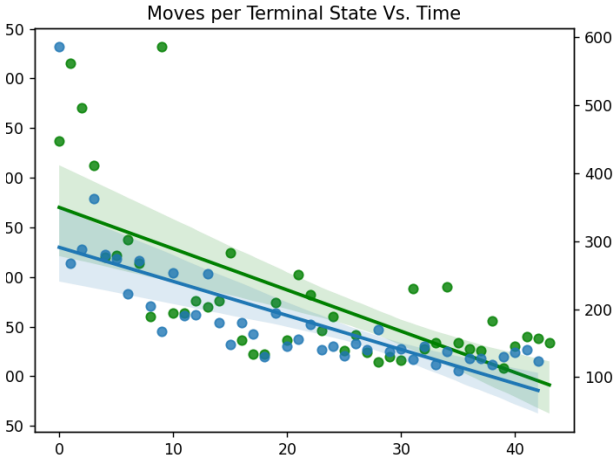

Figure 4.2 Terminal States α = 0.15

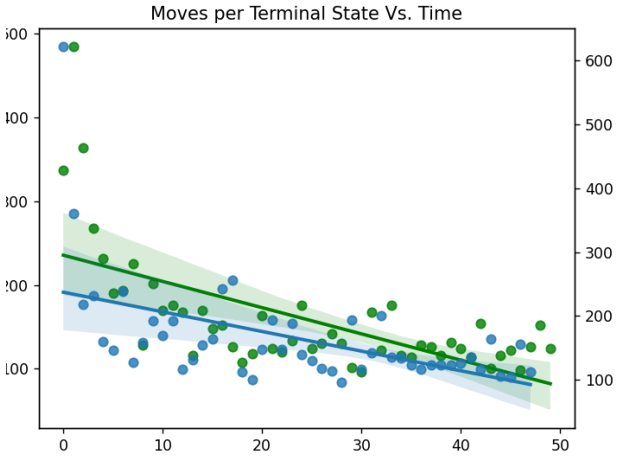

Figure 4.3 Terminal States α = 0.45

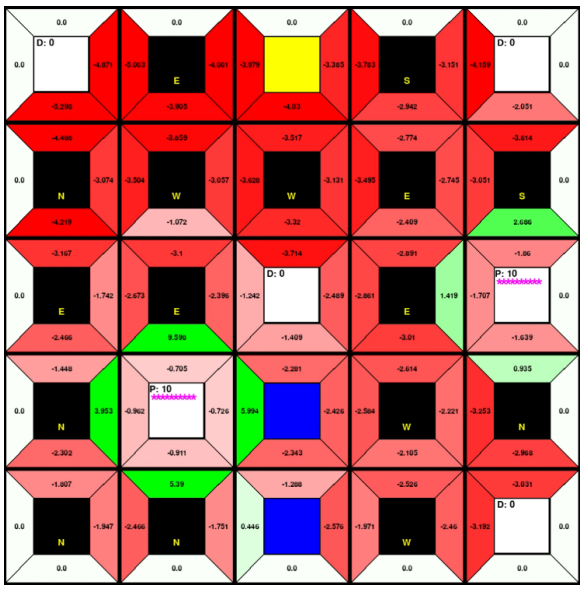

Figure 4.4 Male Qtable (α = 0.15)

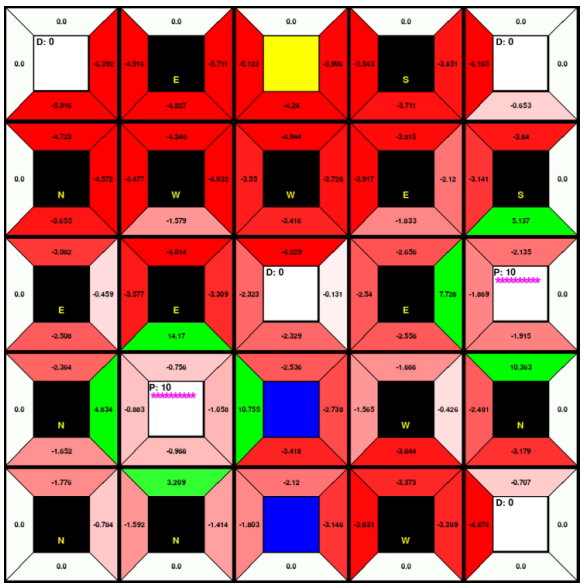

Figure 4.5 Female Qtable (α = 0.15)

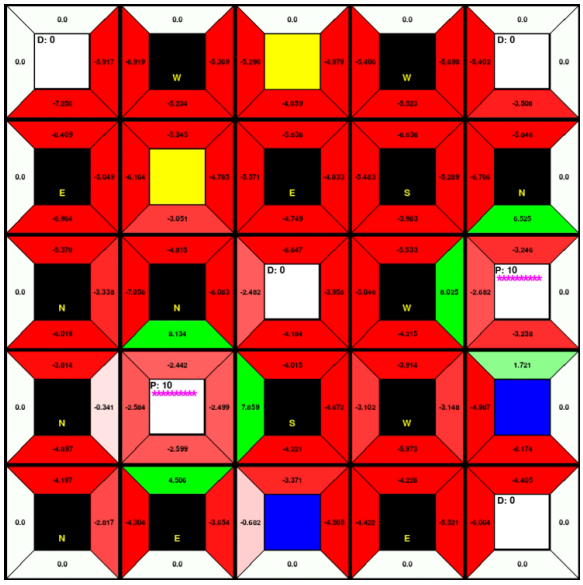

Figure 4.6 Male Qtable (α = 0.45)

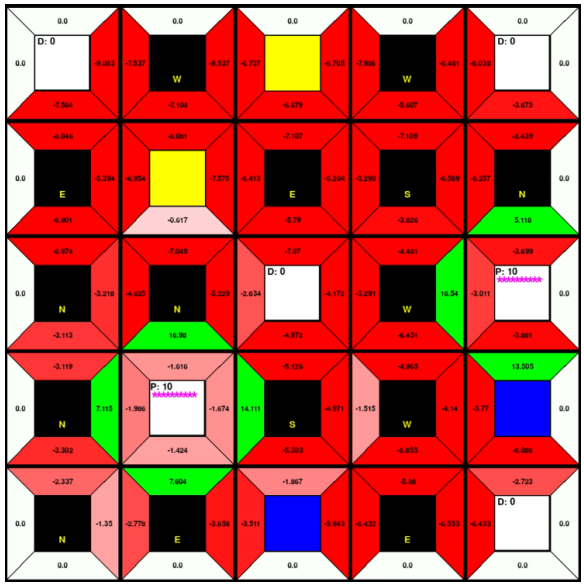

Figure 4.7 Female Qtable (α = 0.45)

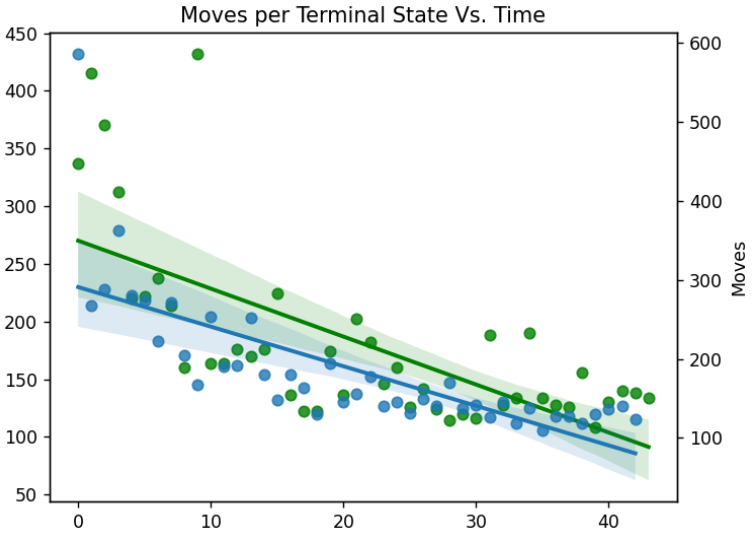

Figure 4.8 Terminal States α = 0.30 (original)

Interpretation

One of the most notable differences between the two learning rates is the number of initial moves to a terminal state and the rate at which this decreases. In the experiment with the learning rate of 0.15, it initially takes approximately 350 moves to

find a terminal state. However, the experiment with learning rate 0.45 takes only 275 moves to find its first terminal state. This fits with the expected behavior of the Q-tables. Initially, all the Q-tables are set to 0, so the lower learning rate will understandably be slower at first because it is putting little weight in new computed utilities despite all utilities being initially to 0. This causes the algorithm to perform suboptimally. On the other hand, the algorithm with the learning rate of 0.45 performed far more successfully, simply because it could more quickly adapt to the environment.

Interestingly, there seems to be a point of diminishing returns where increasing the learning rate does not decrease the number of moves required per terminal state. For instance, the original experiment at $\alpha = 0.30$ also requires roughly 275 moves to reach its first terminal states. Furthermore, looking at the average and minimum number of moves per terminal state in Figure 4.1, the results for $\alpha = 0.30$ and $\alpha = 0.45$ are virtually identical, equally outperforming $\alpha = 0.15$.

The specific point of diminishing return would need to be computed experimentally, for it may occur sooner than $\alpha = 0.3$. However, there seems to be no clear downside to increasing the learning rate up to $\alpha = 0.45$. As a result, we can safely advise increasing the learning rate when applying Q-learning to this style of state space, as it will not adversely affect the results. This may be detrimental however as the agents may be highly susceptible to small changes in the state space and will have trouble defining clear and permanent paths.

Experiment 4 *Noah Gori*

Analyzing strategies ability to adapt and learn/unlearn new/old paths

Procedure

The agent should make 500 moves using PRandom to explore the state space before running to 3 terminal states with PExploit. Pickup Location will then change and the agent will run to 3 terminal states again with PExploit.

Results:

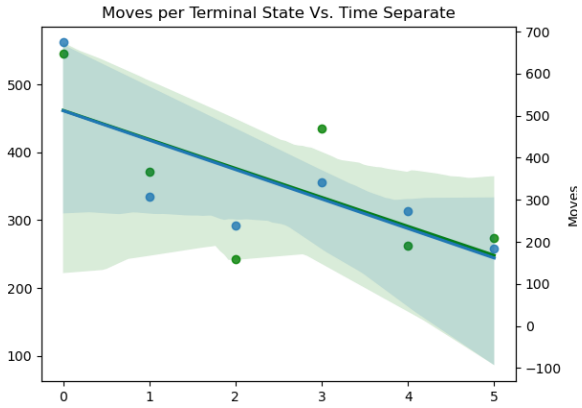

Figure 5.1 Moves Per Terminal State for SARSA and Separate QTables

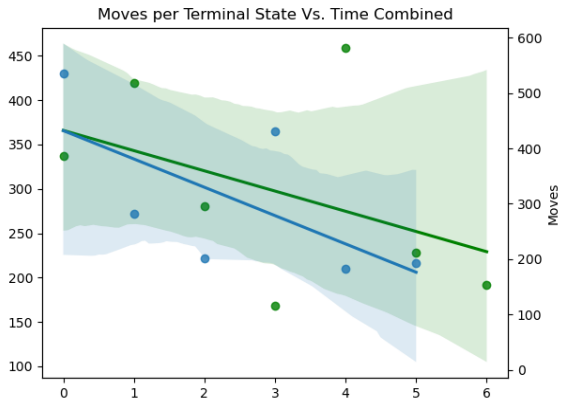

Figure 5.2 Moves Per Terminal State for SARSA and Combined QTables

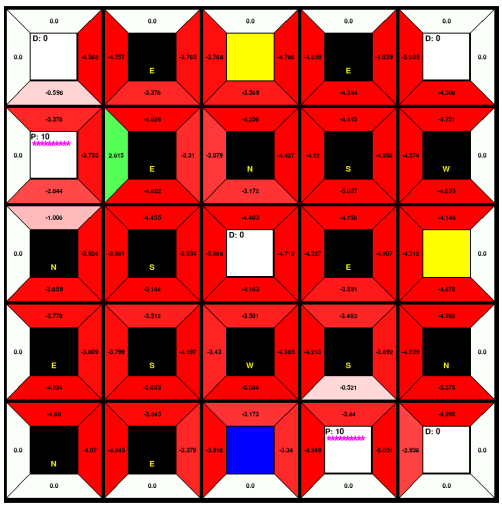

Figure 5.3 Final Q-Values using of female Qtables and X=0

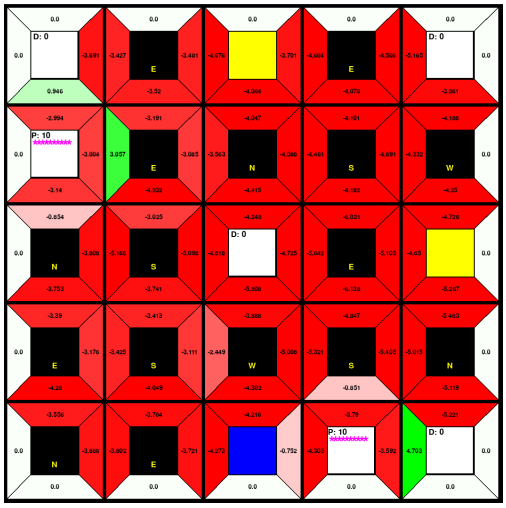

Figure 5.4 Final Q-Values using of male Qtables and X=0

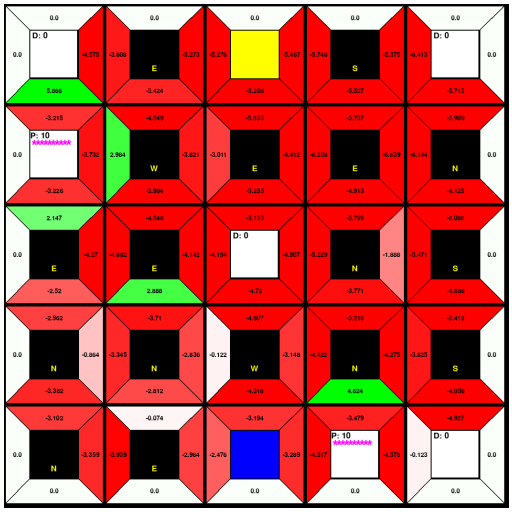

Figure 5.5 Final Q-Values using of combined Qtables and X=0

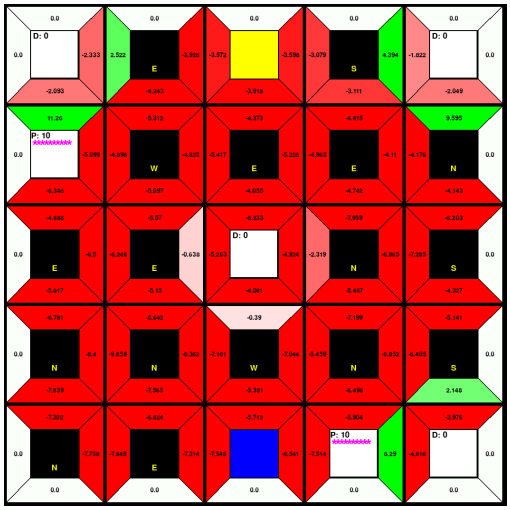

Figure 5.6 Final Q-Values using of combined Qtables and X=1

Average Manhattan Values:

Average Manhattan	Separate - Run1	Separate - Run2	Combined - Run1	Combined - Run2
	3.10925	3.174	3.070875	3.009

Figure 5.7 Average Manhattan Values across runs with SARSA

Interpretation

Figure 5.1 shows that using separate Qtables caused a slightly larger decrease in moves per terminal state which could be due to low blockage as both agents are required to learn their own paths. Figure 5.2 has a larger spread to the MPT values meaning the combined Qtable Runs likely weren't able to develop efficient clear paths to some of the final drop off locations. The largest difference between using the combined and separate QTables that we expected to see was a significant difference in average manhattan difference although the true values we got were around 3.2 and within margin of error of each other. This might be a result of the relatively short time focused on learning in this experiment as opposed to there not being any true difference between the two methods. Comparing figure 5.3 and 5.5 it seems as though the combined QTable was able to generate clearer paths to pick up spaces and for most

locations generated the shortest paths between pickup spots located in the top left and bottom right corners of 1 move of traveling.

Moving on to analysis of whether the learning strategies were able to adapt to changing conditions, judging from the continued downward slope on the graph of both combined and separate MPT values the strategies tend to be generally good at unlearning and relearning new paths. This is likely due to it being necessary to learn new solutions to only half of the problem because once an agent enters the pickup state it can use the previous learned state space for X=1 and follow learned paths.

However it is worth noting that the paths from the female agent location to the nearest pickup spot in the combined QTable was adapted much better than the males as the male still travels to the previous pickup spot or towards the center before moving towards the bottom left pickup spot. Likely this is due to the males proximity to the one of the first set of pickup spots and the large utilities associated with that proximity being more difficult to unlearn. It is also likely that given one or two more terminal states or a larger learning rate it would have similar performance to the female agent.

Looking at figure 5.4 you can actually see that the male agent with separate QTable without the “scrambling” effect of the combined qtable is able to relearn this new path to the pickup spot to the East. The “scrambling” effect may only be present when looking at the final combined QTable because later states will have larger negative utilities due to both agents searching for pickup spots before the pickup spots have changed and masking the initial positive paths towards the new pickup spots. The bottom right section of the board will also have larger negative values that take longer to unlearn due to the pickup spots being focused there in the beginning and agents spending most of the first 3 terminal states there in the X=0 statespace.

Summary

Effectiveness in an artificial agent’s decision comes from the tuning of the foundation it is built on. SARSA is the better choice for our Q-learning algorithm because, as demonstrated by Experiment 2, it allows for greater coordination between the two agents, thus improving the overall performance of the algorithm. In choosing

whether to use the combined or separate QTables for each agent it seems to be beneficial to use the combined tables because the hit in agent coordination did not outweigh benefits of the combined efforts of both agents towards exploring the state space. To aid in the reluctance of the combined QTable to respond to changes of slightly higher learning rate due to slight performance gains and adaptability would be best the choice for a changing PD-World. Given the task at hand, the PGreedy policy served best in finding the optimal solution despite its lack of exploration.