Jacob Bachtarie, Pedro Bavdaz

Professor Eick

Fundamentals of Artificial Intelligence

17 April 2024

PD World Findings

The reinforcement learning system we designed for the “Path Discovery in a 3-Agent Transportation World” project is divided in two classes: agent and world. The environment module is within class world, this module defines the PD-World in a grid-like fashion represented by a matrix. The environment is initialized with parameters: agents, pickup, and dropoff locations within the init_environment method. The reset_environment method is used in certain experiments and when a terminal state is reached to reset the pickup, dropoff, and agent states. Pickup and dropoff locations are also stored within a dictionary for ease of access. Within the class world, the methods pickup() and dropoff() handle the conditionals for an agent to pick up and drop off a block. In addition, to updating the environment to reflect these occurrences. The validOps() function determines the valid operations that can be taken by an agent by gauging the bounds of the environment and the location of other agents. ValidOps() checks if an agent is on a pickup and dropoff location, if the agent is holding a block, and if the location is out of blocks or has reached capacity; these are the conditions for the actions of ‘pick’ and ‘drop’ to be valid. The function randomOp() calls the validOps() function at the beginning to update an agent's operations list, 1 being valid and 0 for an invalid operation. The function only looks at valid operations, with ‘pick’ and ‘drop’ operations holding priority over the random valid action that is chosen otherwise. If there are no valid operations for an agent at a step, then it is skipped; The next agent in order will continue the step. The greedyOp() method functions similarly to randomOp() with a difference in that only actions containing the highest q-value are randomly chosen from, with the ‘pick’ and ‘drop’ actions holding priority. exploitOp() is a mixture of both greedyOp() and randomOp(), where a coin toss is simulated using `random.random()` with an 80% chance of the actions with the highest q-value being chosen and a 20% chance of any valid operation being chosen at random. Like the previous two functions, ‘pick’ and ‘drop’ hold priority. The three functions randomOp(), greedyOp() and exploitOp() represent the policies (PRANDOM, PGREEDY, PEXPLOIT). The functions return either an action picked by the policy or “no valid” in the case of no actions being possible due to an agent's state. The function isTerminal() determines if there are no blocks left in any pickup location and that there are 5 blocks in each dropoff location. The last function in the world class is begin(), this function represents the bulk of the reinforcement learning algorithm by tying in functions of both the agent and world class. It runs for the amount of steps provided as a hyperparameter to the world class, runs the appropriate blocks of code for each experiment depending on the experiment hyperparameter. For specifically experiment 2, when the SARSA q-learning formula is used, a deep copy of the current agent is created with the future state of the current agent. The future action is calculated using the exploit policy, allowing for the q-value for the future state and action to be accessed. Which is passed to the updateQtable function with a bool value to signal that the SARSA q-learning formula should be used. Specifically for experiment 4, at the beginning there’s a condition for if the terminal state has been reached 3 times the pickup locations block values are reset and if the terminal state condition has been reached 6 times the algorithm will end. For experiment 1, there is a sub_experiment hyperparameter to be given to a class world instance to ensure the right policy is used. All sub experiments of experiment 1 use the PRANDOM policy for the first 500 steps. Similarly to experiment 1, for the first 500 steps, the PRANDOM policy is used to determine the actions of agents. After 500 steps, PEXPLOIT is used as the policy for the remaining steps.

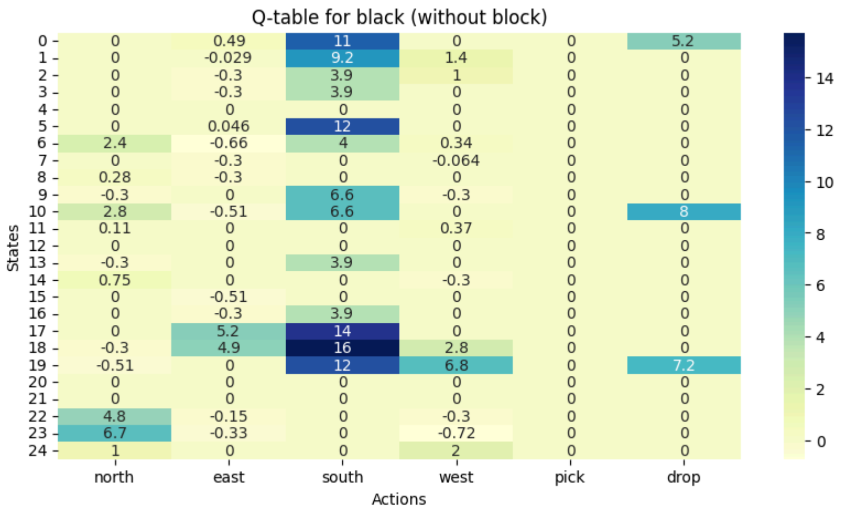
The agent class holds the position of an agent within the environment, if an agent is in possession of a block, the valid actions an agent can take in its current state, learning rate, discount rate and the qtable. Each agent has a qtable of size (25, 2, 6). The first dimension of the q-table represents the locational state, state = row * 5 + col. The second dimension represents if an agent is in possession of a block or not, with 0 representing not being in possession and 1 signifying holding a block. The 3rd dimension represents the possible actions that can be taken by an agent: north, east, south, west, pick, drop. The pick() and drop() functions within the agent class turn the block value to true and false, respectively. The reward() function returns the reward of a given action. The moveNorth(), moveSouth(), moveEast(), moveWest() functions changes an agent's class variables to be used within the environment. The calculate_new_state() function returns the state of an agent that has performed an action given as a hyperparameter. The most important function within the agent class is updateQtable(), it takes the action derived by a policy as a hyperparameter. The other two hyperparameters are if the SARSA q-learning formula is needed for an experiment; A bool value determining if the SARSA q-learning formula should be used, and the q-value of the future state and action chosen by a policy that's needed for the SARSA q-learning formula. The updateQtable() function starts by calculating the reward of the action hyperparameter, determines the new state an agent will be in if said action is taken.

Calculates the max q-value present of any action that can be taken in the current state and uses the traditional q-learning formula provided in the project slides to update the qtable. Using the action variable, it calls the corresponding agent function to update an agent's state. The last function in the agent class is visualizeQtable(), which prints two figures representing an agent's qtable, one with a block and one without. The way to run the program is through calling an instance of the world with the hyperparameters, learning rate, discount rate, number of steps, experiment number, and seed.

Experiments:

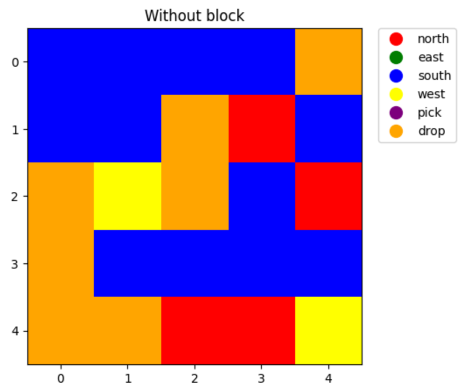
In this section, we present an in-depth analysis of the Q-tables and other information resulting from the experiments we were supposed to run our algorithm through. The Q-tables encapsulate the learned associations between states and actions, providing valuable insights into the behavior and decision-making processes of our agents. We explained briefly how we structured these so that there are 25 states (0-24) which represent all the possible positions on the grig 5 x 5, each one of these columns corresponds to an action to be run at that specific spot, like moving, north, east, south, west, and pick up or drop off. These are different colors and

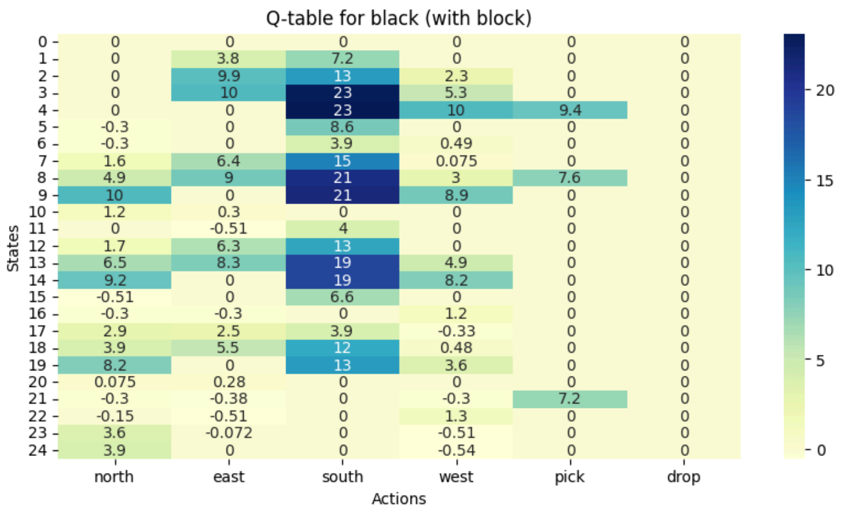
We were tasked with running a few experiments with this algorithm, running with different policies: random exploration (PRANDOM) for 9000, deterministic greedy exploitation (PGREEDY) for 8500 steps, and probabilistic exploitation (PEXPLOIT) for 8500 steps, utilizing values α=0.3 and γ=0.5. This first experiment provided us with the following q tables as product:



Q-table for black (without block)

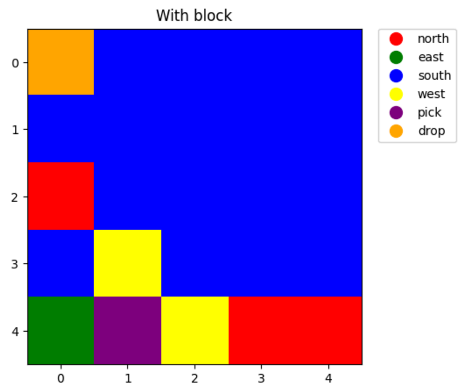
States	north	east	south	west	pick	drop
0	0	0.49	11	0	0	5.2
1	0	-0.029	9.2	1.4	0	0
2	0	-0.3	3.9	1	0	0
3	0	-0.3	3.9	0	0	0
4	0	0	0	0	0	0
5	0	0.046	12	0	0	0
6	2.4	-0.66	4	0.34	0	0
7	0	-0.3	0	-0.064	0	0
8	0.28	-0.3	0	0	0	0
9	-0.3	0	6.6	-0.3	0	0
10	2.8	-0.51	6.6	0	0	8
11	0.11	0	0	0.37	0	0
12	0	0	0	0	0	0
13	-0.3	0	3.9	0	0	0
14	0.75	0	0	-0.3	0	0
15	0	-0.51	0	0	0	0
16	0	-0.3	3.9	0	0	0
17	0	5.2	14	0	0	0
18	-0.3	4.9	16	2.8	0	0
19	-0.51	0	12	6.8	0	7.2
20	0	0	0	0	0	0
21	0	0	0	0	0	0
22	4.8	-0.15	0	-0.3	0	0
23	6.7	-0.33	0	-0.72	0	0
24	1	0	0	2	0	0





Q-table for black (with block)

States	north	east	south	west	pick	drop
0	0	0	0	0	0	0
1	0	3.8	7.2	0	0	0
2	0	9.9	13	2.3	0	0
3	0	10	23	5.3	0	0
4	0	0	23	10	9.4	0
5	-0.3	0	8.6	0	0	0
6	-0.3	0	3.9	0.49	0	0
7	1.6	6.4	15	0.075	0	0
8	4.9	9	21	3	7.6	0
9	10	0	21	8.9	0	0
10	1.2	0.3	0	0	0	0
11	0	-0.51	4	0	0	0
12	1.7	6.3	13	0	0	0
13	6.5	8.3	19	4.9	0	0
14	9.2	0	19	8.2	0	0
15	-0.51	0	6.6	0	0	0
16	-0.3	-0.3	0	1.2	0	0
17	2.9	2.5	3.9	-0.33	0	0
18	3.9	5.5	12	0.48	0	0
19	8.2	0	13	3.6	0	0
20	0.075	0.28	0	0	0	0
21	-0.3	-0.38	0	-0.3	7.2	0
22	-0.15	-0.51	0	1.3	0	0
23	3.6	-0.072	0	-0.51	0	0
24	3.9	0	0	-0.54	0	0

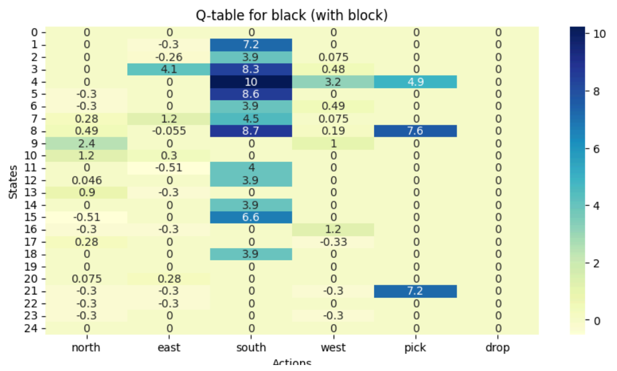


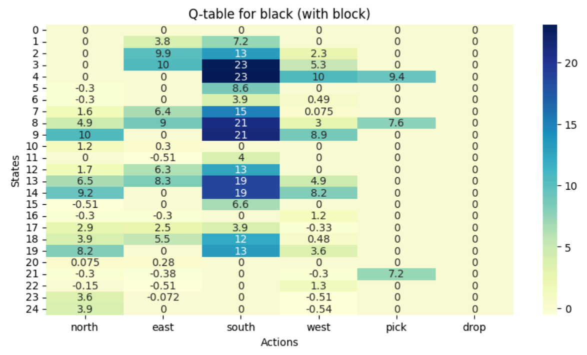
QTables on the left and a “heat map” of the preferred choice in each cell of the 5x5

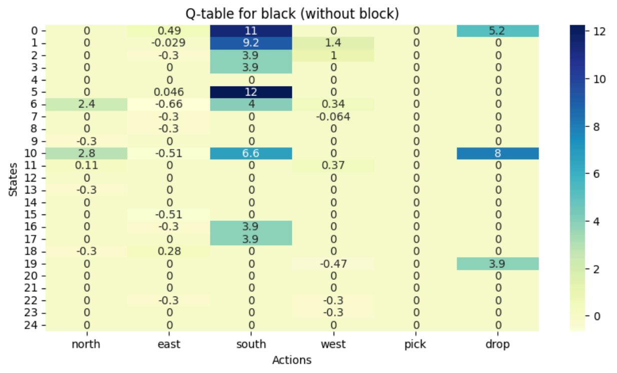
This is one of the resulting tables for experiment 1c. As we can see by the end of the run, in this case, our black agent is able to locate the pickup location found at the bottom left of the world, shown by the purple cell, when not holding a block, and in the other case is also able to find at least 2 of the drop zones, one on the top left and on the left bottom shown with orange cells. It seems to develop a bias to go south on all of our experiments/runs, obviously only applying to our greedy, and exploit policies, not the random. This could be due to a lot of different factors, mainly we believe it's because in the case of our blue agent, starts at the bottom of the world. This makes it so that this particular agent tends to have a better understanding of the bottom or south of the 5x5 world, causing it to lean more on the south side for finding rewards, compared to exploring north which would be penalized with -1 from moving, making the agent stay at the bottom. Or in the case of this black agent, would be finding a single path that works and sticking with it even if it means missing out on closer rewards due to how we have structured our algorithm.

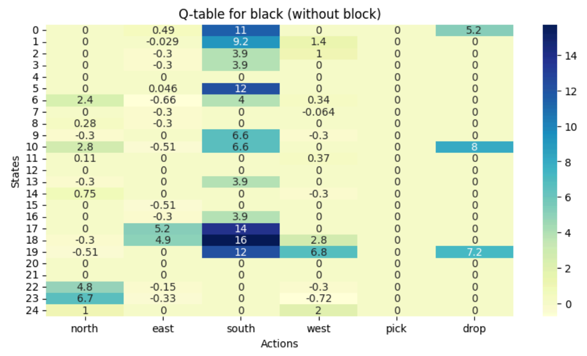
Experiment 1B

Experiment 1C



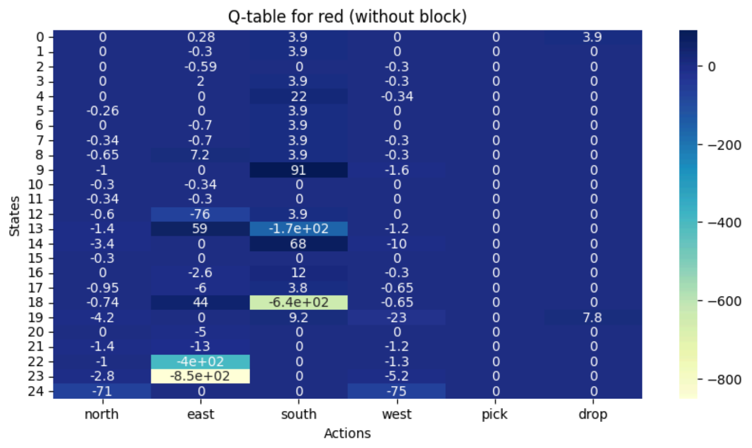


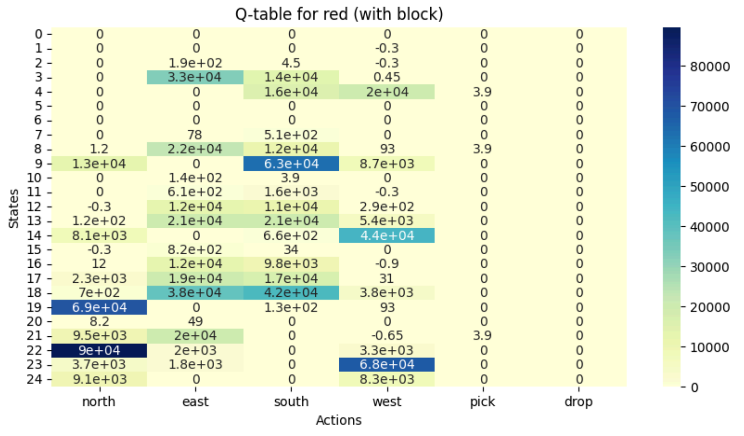




Comparing these two we can see a lot of similarities, not necessarily movement-wise, but both had a pretty solid idea of where the drop-off and pick-up locations were located with experiment 1B having slightly higher values in those specific spots. We can find a lot of differences in the movement, still both favoring going south, but experiment 1C tends to wonder a bit more around especially east and west when possessing a block this is probably due to the fact that all drop zones are in the edges of the world, and by searching further to the sides it's able to locate them, and so it has the higher q values in the correct pickup and drop spots. Overall both have pretty similar patterns, with 1C having slightly more variation, allowing it to solidify the drop-off and pick-up locations more efficiently.

Experiment two:

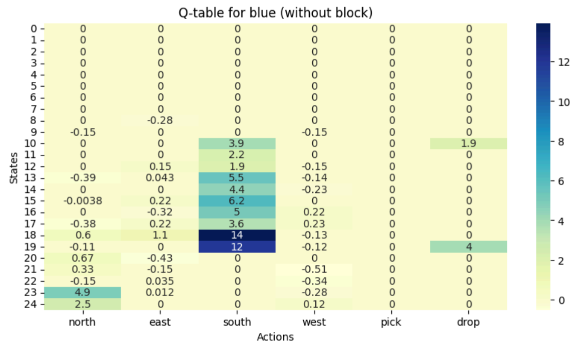


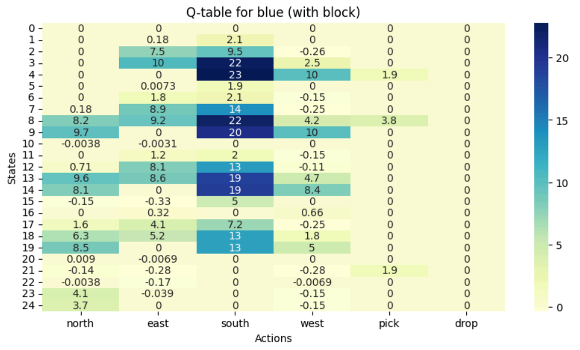


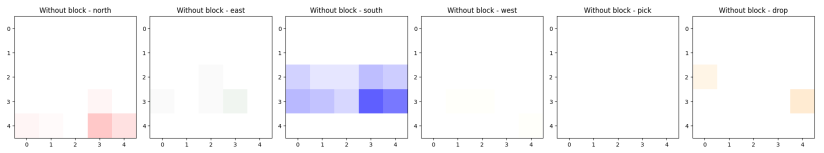
For experiment two we ran out algorithm utilizing our implementation of the SARSA q-learning variation. As you can see from the QTables, SARSA wasn't fully or at least correctly implemented, though it was able to find some of the drop-off, and pick-up locations, as seen by the low values in the pick and drop column, the functions seem to give us subpar results. This is shown by the abnormally high numbers found in the q table. All the agents in this run gave very different results which were very volatile since they would change a lot from run to run. In our case, the Q-Learning algorithm worked a lot better, compared to these results.

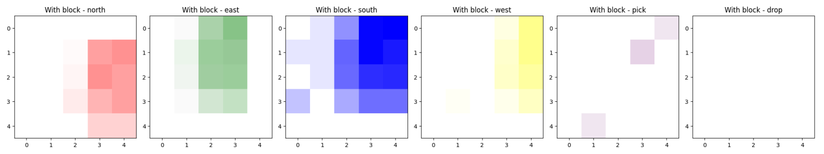
Experiment Three:

2 with Learning Rate 0.15

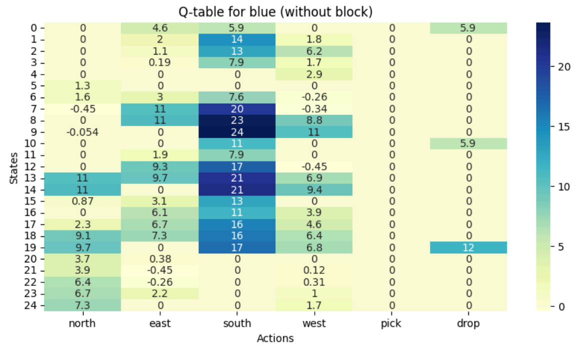


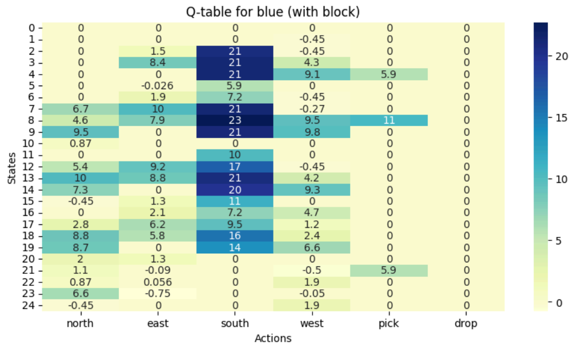


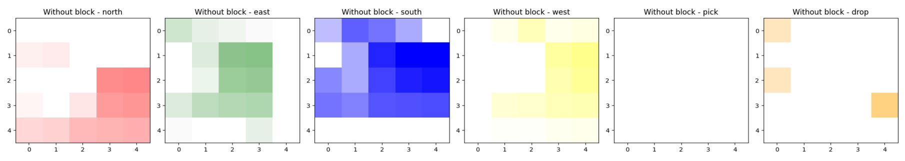


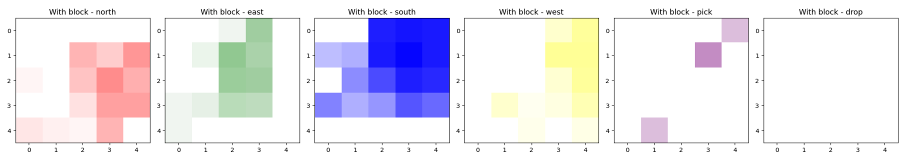


2 with Learning Rate 0.45





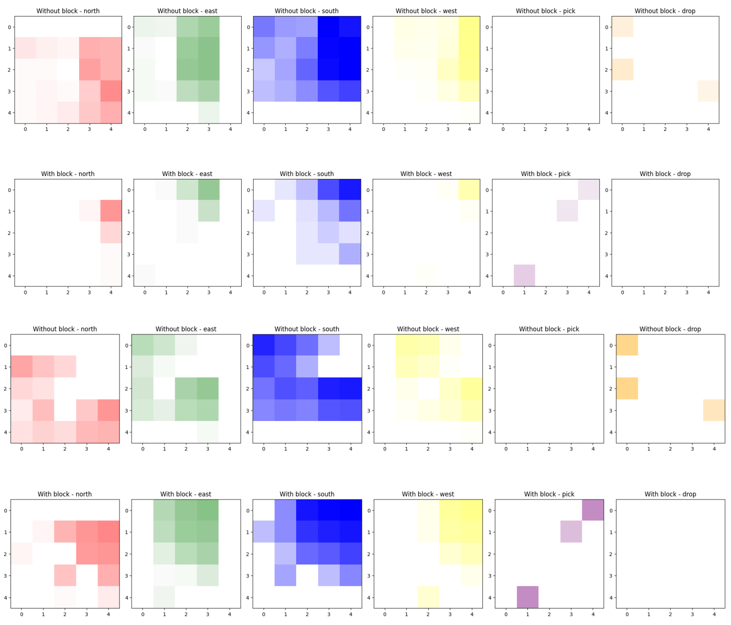




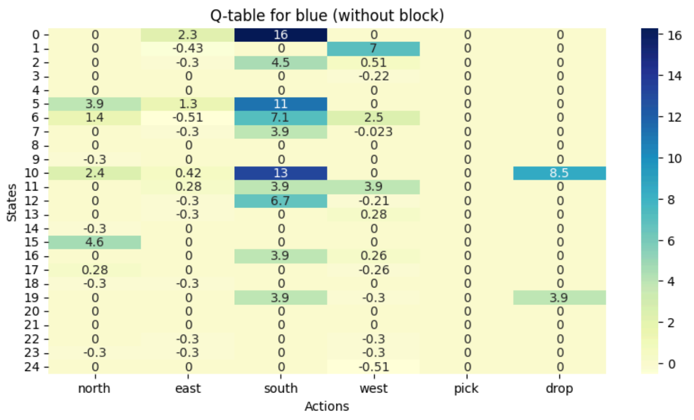
QTables (top), heat map for each preferable action on the 5x5

As we can see in experiment 3 we can see more activity, in this case on the blue agent, which helps him quite a lot. We can see in both sets of data how much more defined the higher q value the dropoff and pick-up areas have, showing a higher understanding of the world, in a shorter time frame, so much that in the first example of 0.15 LR the blue agent since it starts at the bottom, doesn't even explore up, staying on the southern part of the world due to the knowledge it's acquired, staying with the most rewarding option. This is not necessarily a good thing, in this case, it didn't, but with a sufficiently large LR the agent could have been "blinded" making one action too favorable. This would make it so that instead of exploring further, it could find a route that might not be the best one but will stick with it because of the skewed or overweight q values produced. We don't really see that in ours. In this case, it seemed to help the agent a lot, allowing it to learn the dropoff and pick-up locations faster.

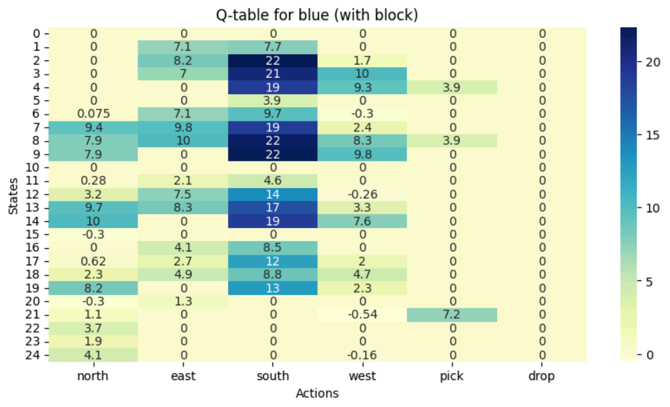
We can see the same changes on other agents in this case Red, with the higher LRs leading to higher q values (color strength) in the drop-offs and pick-ups. As seen here, this shows an overall improvement in the ability to find the locations for all agents alike.



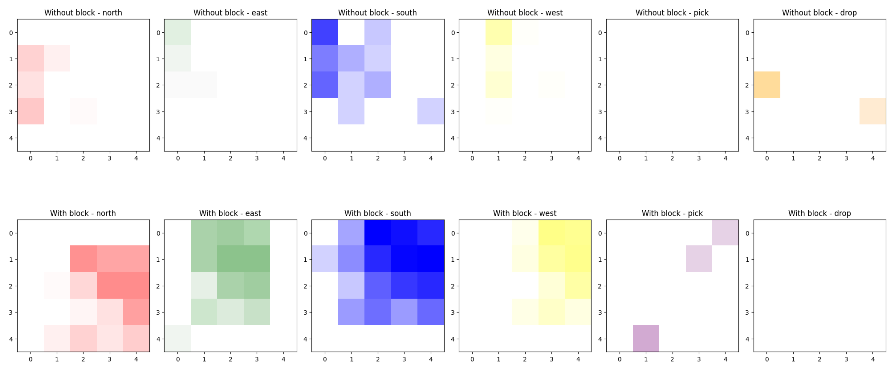
Experiment 4:



States	north	east	south	west	pick	drop
0	0	2.3	16	0	0	0
1	0	-0.43	0	7	0	0
2	0	-0.3	4.5	0.51	0	0
3	0	0	0	-0.22	0	0
4	0	0	0	0	0	0
5	3.9	1.3	11	0	0	0
6	1.4	-0.51	7.1	2.5	0	0
7	0	-0.3	3.9	-0.023	0	0
8	0	0	0	0	0	0
9	-0.3	0	0	0	0	0
10	2.4	0.42	13	0	0	8.5
11	0	0.28	3.9	3.9	0	0
12	0	-0.3	6.7	-0.21	0	0
13	0	-0.3	0	0.28	0	0
14	-0.3	0	0	0	0	0
15	4.6	0	0	0	0	0
16	0	0	3.9	0.26	0	0
17	0.28	0	0	-0.26	0	0
18	-0.3	-0.3	0	0	0	0
19	0	0	3.9	-0.3	0	3.9
20	0	0	0	0	0	0
21	0	0	0	0	0	0
22	0	-0.3	0	-0.3	0	0
23	-0.3	-0.3	0	-0.3	0	0
24	0	0	0	-0.51	0	0



States	north	east	south	west	pick	drop
0	0	0	0	0	0	0
1	0	7.1	7.7	0	0	0
2	0	8.2	22	1.7	0	0
3	0	7	21	10	0	0
4	0	0	19	9.3	3.9	0
5	0	0	3.9	0	0	0
6	0.075	7.1	9.7	-0.3	0	0
7	9.4	9.8	19	2.4	0	0
8	7.9	10	22	8.3	3.9	0
9	7.9	0	22	9.8	0	0
10	0	0	0	0	0	0
11	0.28	2.1	4.6	0	0	0
12	3.2	7.5	14	-0.26	0	0
13	9.7	8.3	17	3.3	0	0
14	10	0	19	7.6	0	0
15	-0.3	0	0	0	0	0
16	0	4.1	8.5	0	0	0
17	0.62	2.7	12	2	0	0
18	2.3	4.9	8.8	4.7	0	0
19	8.2	0	13	2.3	0	0
20	-0.3	1.3	0	0	0	0
21	1.1	0	0	-0.54	7.2	0
22	3.7	0	0	0	0	0
23	1.9	0	0	0	0	0
24	4.1	0	0	-0.16	0	0



As we can see from the graphs the algorithm wasn't able to adapt to the change of pick-up locations at all. This could be because of multiple reasons, mainly because of the algorithm already having learned the old layout and relying on it even after the change, since none of the new pick-up locations are near the old ones, the agents would have to take way fewer rewarding paths than the ones they already had set. This caused what we can see above which is a little less exact finding for the drop-off and pick-up locations but it's still very much the old locations. Also, the way our algorithm works, in the area of the new pickups, there is nothing aside from a drop-off, so our agents, if they are not currently carrying anything on them, would go through the better path leading to the original pick up areas, completely avoiding the new pick up location.

In our algorithm, the most successful experiment was number 3, showcasing the different learning rates. With the heatmaps it's very easy to see the pickup and drop off location areas being highlighter by the higher QTable number present, and the more “harsh decisions”, especially when dropping off, it being more prone to exploring west and east since the locations are at the edges of the world. We can also find many paths or strategies in some of these agents, in the example of experiment number 3, we see the blue agent staying on the left side of the world, near the dropoff location, but when emptyhanded it tends to move more to the left side of the world. The only time our algorithm was able to achieve the terminal state was in experiment 1, achieving a top 7 termination states. This could be due to the use of a mixture of different policies, allowing for more randomized ways of searching. This, in turn, would make the algorithm explore the world a little better, lowering the chances of it sticking with undesirable paths.