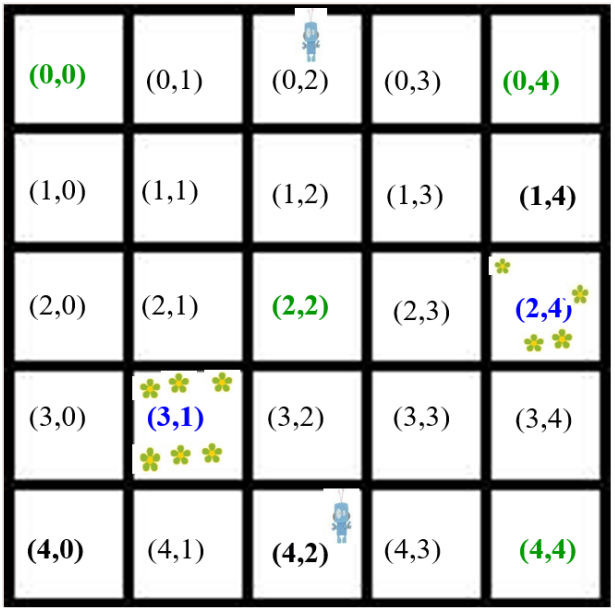
Using Reinforcement Learning to discover paths in a 2-Agent Transportation World problem

Group Mimosa
Jennifer Csicsery-Ronay PSID: 1922775
Arno Dunstatter PSID: 1926976
Kevin Hsieh PSID: 2078611
Jinangkumar Shah PSID: 2034013

In this project we utilized Q-learning/SARSA reinforcement learning to learn and adapt “promising paths” in a 2-agent setting described below in *Figure 1.*

Figure 1



Agent F is in cell (0,2)
Agent M is in cell (4,2)
Pickup cells are (3,1) and (2,4) and each contain 10 ✿ initially
Dropoff cells are (0,0), (0,4), (2,2), and (4,4) and can hold up to 5 ✿
Terminal state: all **Dropoff cells** are full, containing 5 ✿
Goal: Transport ✿ from **Pickup** to **Dropoff** to reach a terminal state

Two agents named ‘M’ (male) and ‘F’ (female) are solving the block transportation problem jointly. Agents alternate applying operators to the PD-World, with the female agent acting first. Moreover, both agents cannot be in the same position at the same time; consequently, there is a blockage problem, limiting agent mobility and ultimately efficiency in cases where both agents are attempting to work on the same path at the same time. Four experiments are conducted with different parameters and three different policies. When Q-learning/SARSA was performed we created what’s called a Q-table that in theory can be represented as a matrix with the shape of [state, action] where all Q-values therein are initialized to zero. In practice, both to reduce the number of rows in our Q-table and only allow for positionally applicable movements we utilized a more complex data structure described later. We then update and store our q-values after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value. During an episode, each of the agents alternate turns (with the female agent acting first in the very first episode) and selects one action from positionally applicable operators, based on the policy presently in use.

Reinforcement Learning Theory

Overview

To fully grasp the concept of reinforcement learning, we first must understand how q-learning works and how it is different from SARSA. To begin, we first introduce some definitions of influence variables and functions of our design and implementation of the algorithms.

State *s*: represents the current configuration of the world and the agents therein. What information is embedded in each state is up to the designer of the state-space but can include variables such as agents' positions, their holding statuses, the number of blocks in each position, whether particular pickup locations still contain blocks (or how many they contain), whether particular dropoff locations still can accept blocks (or how many they can still accept), etc.

Action *a*: the step taken by the agent when it is in a particular state.

State *s'*: next state to which the agent goes from State *s.*

Action *a':* next action to which the agent chooses in State *s'.*

Discount factor γ (gamma): it quantifies how much importance we give for future rewards. It's also handy to approximate the noise in future rewards. Gamma varies from 0 to 1. If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.[3]

Learning rate α (alpha): controls how quickly the robot adopts to the random changes imposed by the environment.

Rewards, *R(s, a)*: for every action, the agent will get a positive or negative reward.

Q-Values, *Q (a, s)*: used to determine how good an Action, *a*, taken at a particular state, *s*, is.

Q-learning update function: $Q(s_t, a_t) = Q(s_t, a_t) + \alpha * [R_t + \gamma * max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$

SARSA update function:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * [R_t + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

where R_t is the reward received from the environment at timestep *t*. [1]

Design and implementation

Q-table: a reference table containing presently expected future rewards for all possible actions at each state in the state space. For policies which utilize the Q-table it serves to guide the agent to choose an action which is presently understood to be the best for the particular state the agent is presently in. All Q-values (utility values) are initialized to zero. In most theoretical descriptions of Q-tables they are represented as a matrix with every row representing a particular state and every column representing an action, however this results in very long Q-tables where every state has the same number of actions to choose from. In our actual implementation we chose a different data structure to represent our Q-table, both to reduce its length and to only allow positionally applicable actions (i.e. the agent doesn't even have the option to move north if it's already at the top of the board). This data structure is a dictionary of dictionaries of lists of lists. The outermost dictionary has tuples for keys which represent positions on the gameboard and values which are themselves dictionaries. These inner dictionaries have keys which are movement-action strings ("north", "south", "east", "west") and values which are lists of lists with the outer list being accessible based on a bool representing whether the presently acting agent is holding a block. If the agent is holding (i.e. holding=True) then the inner list has a length of 16 and the indexing represents all possible combinations of whether the four dropoff locations can still accept more blocks (2^4). If the agent is not holding (i.e. holding=False) then the inner list has a length of 4 and the indexing represents all possible combinations of whether the two pickup locations still have blocks (2^2). Only actions which are applicable at each particular position are included as keys for that position's inner dictionary.

State Space: In our implementation a particular state is composed of the following variables: the row, and column of the presently acting agent, a boolean value representing whether this presently acting agent is already holding a block, and an index representing either a particular combination of pickup cells still having blocks (if the presently acting agent is not holding) or a particular combination of dropoff cells still being able to accept blocks (if the presently acting agent is holding). Size of state space will be around 25(total locations) * 2(holding o not holding) * 5(maximum possible action for each possible location) * 4(different pick up states) * 16(different drop off states) = 16,000.

Advantage: No need to unlearn paths; as no unlearning using this state space occurs the team of agents might get more efficient in later runs after already solving the transportation problem multiple times.

Disadvantage: Space is 64 times larger than other reduced state space such as Space1 from the problem statement provided.

Policies π: provides the guideline on what is the optimal action to take in a certain state with the goal to maximize the total rewards.

- **PRANDOM policy:** If pickup and dropoff is applicable, choose this operator; otherwise, choose an operator randomly.
- **PEXPLOIT policy:** If pickup and dropoff is applicable, choose this operator; otherwise, apply the applicable operator with the highest q-value (break ties by rolling a dice for operators with the same utility) with probability 0.80 and choose a different applicable operator randomly with probability 0.20.
- **PGREEDY policy:** If pickup and dropoff is applicable, choose this operator; otherwise, apply the applicable operator with the highest q-value (break ties by rolling a dice for operators with the same utility).

Operators: mappings from one point to another within the vector space of state values, $\mathbb{R}^n$, where $n = |S|$ is the size of the state space and we define an operator T^{π}: $\mathbb{R}^n \rightarrow \mathbb{R}^n \; as \; T^{\pi}(v) = R^{\pi} + \gamma P^{\pi} v$ for any $v \in \mathbb{R}^n$

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\pi}(s') \right)$$

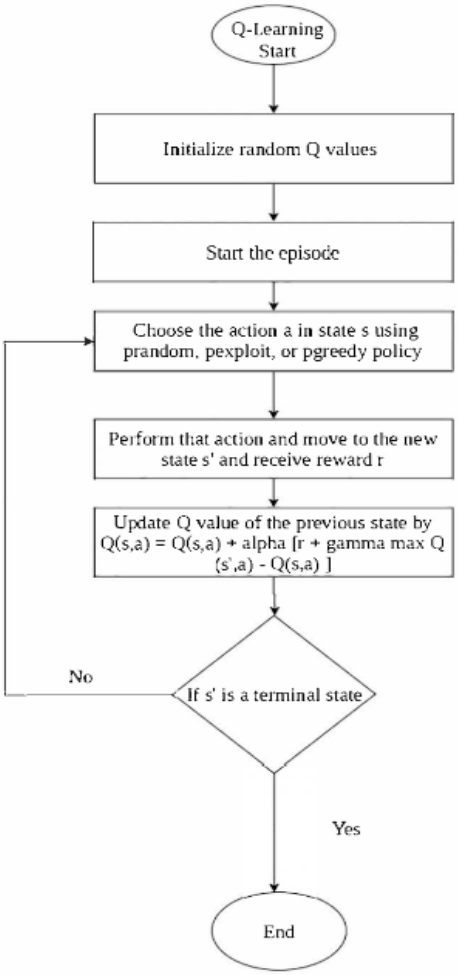
from the Bellman equation:[2]

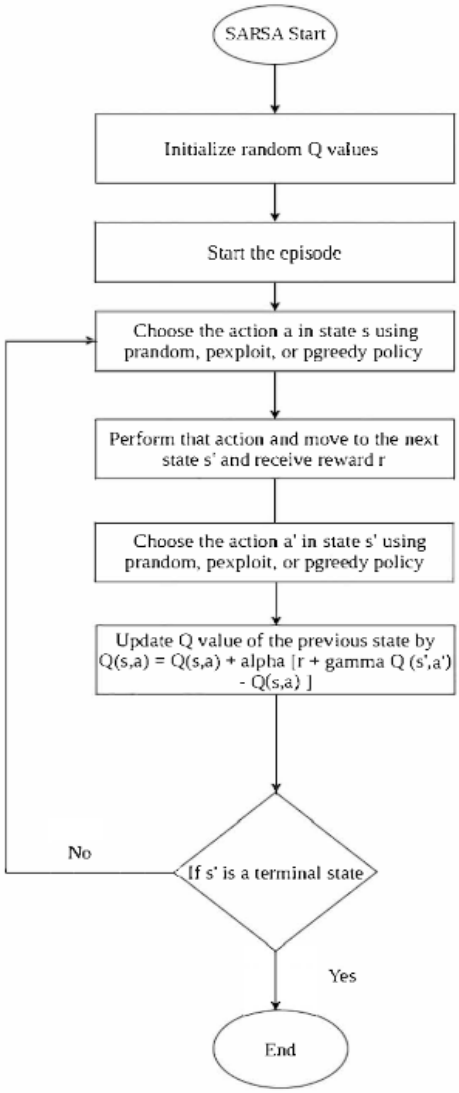
For this project, we have a total of 6 operators: north, south, east, west, pickup, and dropoff, only one operator (action) may be applied by an agent per turn.

- **North, South, East, West**: applicable in each state, and moving the agent to the cell in that direction except leaving the grid is not allowed.
- **Pickup:** only applicable if the agent is in a pickup cell that contains at least one block and if the agent does not already carry a block.
- **Dropoff:** only applicable if the agent is in a dropoff cell that contains less than 4 blocks and the agent carries a block.

Operators (actions) are something a reinforcement learning agent can do to change states *s*. And rewards are the utility the agent receives for performing the “right” actions.

Comparison between q-learning and SARSA





In both methods we take an action *a* from state *s* to a new state *s’*, observing a reward *r*. The action *a* is selected by following the current policy and thereafter we update *Q(s, a)*: in SARSA, this is done by choosing another action *a’* following the same current policy and using $r + \gamma Q(s', a')$ as the target. SARSA is called on-policy learning because it updates $Q(a, s)$ using the action a' chosen by the *current policy* in state s'. Whereas in Q-learning $Q(a, s)$ is updated using an estimate of $Q(a', s')$, $max_{a'} Q(a', s')$, i.e. the action a' chosen by the greedy policy may not actually be the action selected at s'.[4, 5]

Experiments & Interpretation

For all experiments, if the agent reaches a terminal state, we restart the experiment by resetting the PD world to the initial state, but not the Q-table. Each experiment was run twice with different seeds and was compared and assessed below to see which had the better result. The table below shows the parameters we chose for our experiments.

Experiment	Algorithm	Learning Rate	Policy	Steps	Actions to Take Once Termination Is Reached
1	Q-Learning	0.3	PRANDOM	500	reset PD world to initial state
1-a	Q-Learning	0.3	PRANDOM	7500	reset PD world to initial state
1-b	Q-Learning	0.3	PGREEDY	7500	reset PD world to initial state
1-c	Q-Learning	0.3	PEXPLOIT	7500	reset PD world to initial state
2	SARSA	0.3	PRANDOM	500	reset PD world to initial state
	SARSA	0.3	PEXPLOIT	7500	reset PD world to initial state
3-a	Q-Learning	0.15	PRANDOM	500	reset PD world to initial state
	Q-Learning	0.15	PEXPLOIT	7500	reset PD world to initial state
3-b	Q-Learning	0.45	PRANDOM	500	reset PD world to initial state
	Q-Learning	0.45	PEXPLOIT	7500	reset PD world to initial state
4	Q-Learning	0.3	PRANDOM	500	reset PD world to initial state
	Q-Learning	0.3	PEXPLOIT		run until 3 terminal states reached, then change pickup locations to (0,1), (3,4), continue running PEXPLOIT without resetting Q-table until 6 terminal states

Q-Tables

Our Q-tables for the relevant runs were rather large due to our implementation of a state space similar to state space 2 given in the powerpoint covering the assignment. To examine the Q-tables in their full detail please see the files in the relevant folder.

Results

Raw results

Hivemind: True, seed = 577

```
Experiment 1a took 1.2297016530000002s
Terminal States reached: 6
Steps   per terminal state: [1367, 1007, 1207, 1076, 912, 2039]
Rewards per terminal state: [-863, -503, -703, -572, -408, -1535]



Experiment 1b took 1.2085029930000002s
Terminal States reached: 35
Steps   per terminal state: [999, 459, 324, 256, 183, 172, 232, 196, 307, 175, 184, 279, 167, 256, 247, 171, 176, 228, 176, 172, 179, 167, 172, 176, 172, 188, 172, 172, 172, 176, 172, 199, 167, 172, 172]
Rewards per terminal state: [-495, 45, 180, 248, 321, 332, 272, 308, 197, 329, 320, 225, 337, 248, 257, 333, 328, 276, 328, 332, 325, 337, 332, 328, 332, 316, 332, 332, 332, 328, 332, 305, 337, 332, 332]



Experiment 1c took 1.3124156619999998s
Terminal States reached: 29
Steps   per terminal state: [1068, 384, 432, 295, 243, 224, 227, 251, 203, 231, 207, 363, 188, 199, 220, 279, 200, 239, 219, 228, 188, 215, 184, 215, 239, 223, 247, 220, 227]
Rewards per terminal state: [-564, 120, 72, 209, 261, 280, 277, 253, 301, 273, 297, 141, 316, 305, 284, 225, 304, 265, 285, 276, 316, 289, 320, 289, 265, 281, 257, 284, 277]



Experiment 2 took 1.2076650549999997s
Terminal States reached: 26
Steps   per terminal state: [991, 407, 359, 496, 372, 335, 271, 220, 232, 231, 212, 215, 255, 228, 224, 227, 300, 235, 343, 264, 239, 248, 323, 219, 211, 212]
Rewards per terminal state: [-487, 97, 145, 8, 132, 169, 233, 284, 272, 273, 292, 289, 249, 276, 280, 277, 204, 269, 161, 240, 265, 256, 181, 285, 293, 292]



Experiment 3 with alpha=0.15 took 1.2504433619999995s
Terminal States reached: 26
Steps   per terminal state: [1128, 479, 440, 395, 288, 347, 320, 376, 256, 251, 220, 200, 216, 259, 212, 219, 235, 247, 192, 199, 275, 199, 255, 212, 203, 203]
Rewards per terminal state: [-624, 25, 64, 109, 216, 157, 184, 128, 248, 253, 284, 304, 288, 245, 292, 285, 269, 257, 312, 305, 229, 305, 249, 292, 301, 301]



Experiment 3 with alpha=0.45 took 1.2546076619999997s
Terminal States reached: 29
Steps   per terminal state: [940, 559, 324, 363, 315, 223, 263, 199, 243, 231, 239, 224, 199, 231, 264, 183, 211, 208, 236, 223, 215, 200, 215, 220, 227, 191, 244, 212, 195]
Rewards per terminal state: [-436, -55, 180, 141, 189, 281, 241, 305, 261, 273, 265, 280, 305, 273, 240, 321, 293, 296, 268, 281, 289, 304, 289, 284, 277, 313, 260, 292, 309]



Experiment 4 took 0.3471403360000007s
Terminal States reached: 6
Steps   per terminal state: [1068, 384, 432, 644, 583, 499]
Rewards per terminal state: [-564, 120, 72, -140, -79, 5]
```

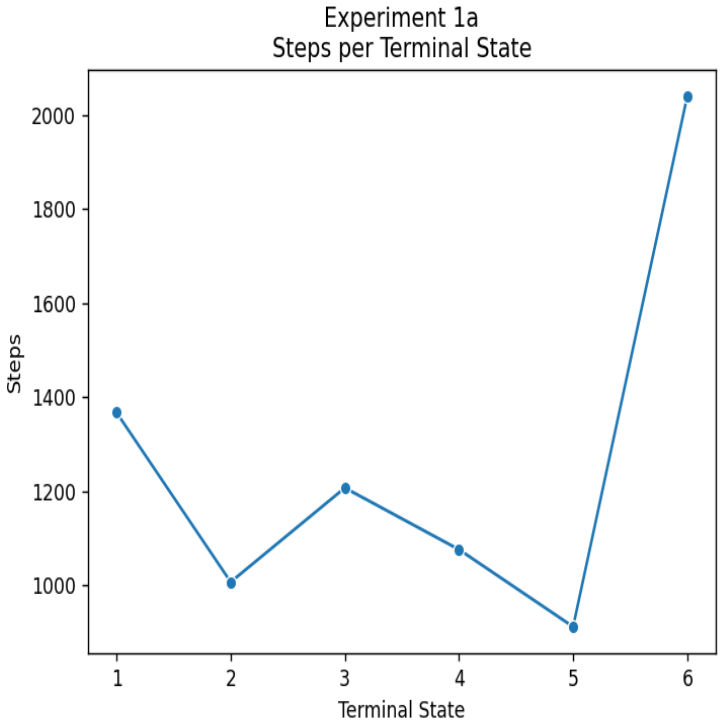
Hivemind = False, seed = 326

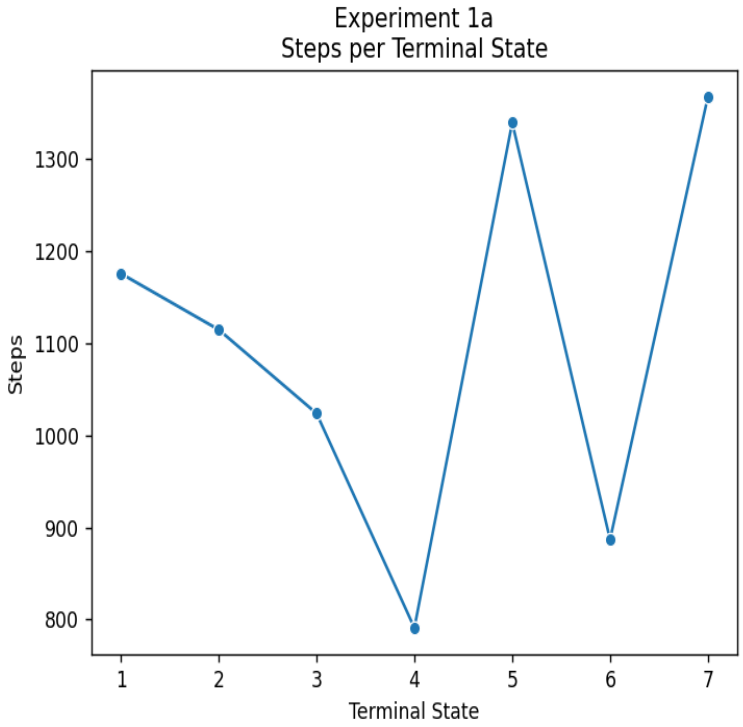
```
Experiment 1a took 1.242594522s
Terminal States reached: 7
Steps   per terminal state: [1176, 1115, 1024, 791, 1340, 887, 1368]
Rewards per terminal state: [-672, -611, -520, -287, -836, -383, -864]



Experiment 1b took 1.221696236s
Terminal States reached: 37
Steps   per terminal state: [999, 399, 443, 260, 204, 220, 219, 188, 187, 187, 179, 179, 172, 168, 192, 168, 168, 168, 168, 168, 168, 168, 183, 203, 180, 180, 168, 168, 168, 168, 168, 168, 168, 168, 168, 172, 176]
Rewards per terminal state: [-495, 105, 61, 244, 300, 284, 285, 316, 317, 317, 325, 325, 332, 336, 312, 336, 336, 336, 336, 336, 336, 336, 321, 301, 324, 324, 336, 336, 336, 336, 336, 336, 336, 336, 336, 332, 328]



Experiment 1c took 1.2837109830000002s
Terminal States reached: 29
Steps   per terminal state: [951, 584, 339, 508, 292, 232, 203, 208, 220, 220, 212, 207, 263, 227, 220, 228, 220, 220, 195, 232, 235, 251, 176, 207, 219, 244, 212, 259, 199]
Rewards per terminal state: [-447, -80, 165, -4, 212, 272, 301, 296, 284, 284, 292, 297, 241, 277, 284, 276, 284, 284, 309, 272, 269, 253, 328, 297, 285, 260, 292, 245, 305]



Experiment 2 took 1.2153131789999998s
Terminal States reached: 26
Steps   per terminal state: [944, 420, 380, 328, 328, 439, 243, 200, 247, 520, 204, 327, 271, 251, 200, 271, 208, 232, 223, 248, 227, 235, 291, 219, 251, 184]
Rewards per terminal state: [-440, 84, 124, 176, 176, 65, 261, 304, 257, -16, 300, 177, 233, 253, 304, 233, 296, 272, 281, 256, 277, 269, 213, 285, 253, 320]



Experiment 3 with alpha=0.15 took 1.2745121630000007s
Terminal States reached: 24
Steps   per terminal state: [984, 420, 380, 428, 435, 524, 319, 316, 268, 368, 287, 348, 244, 239, 307, 251, 224, 227, 216, 215, 240, 256, 228, 220]
Rewards per terminal state: [-480, 84, 124, 76, 69, -20, 185, 188, 236, 136, 217, 156, 260, 265, 197, 253, 280, 277, 288, 289, 264, 248, 276, 284]



Experiment 3 with alpha=0.45 took 1.2609388419999998s
Terminal States reached: 27
Steps   per terminal state: [995, 639, 423, 252, 263, 211, 211, 255, 256, 211, 239, 231, 224, 235, 231, 220, 291, 248, 235, 232, 216, 300, 396, 208, 260, 279, 215]
Rewards per terminal state: [-491, -135, 81, 252, 241, 293, 293, 249, 248, 293, 265, 273, 280, 269, 273, 284, 213, 256, 269, 272, 288, 204, 108, 296, 244, 225, 289]



Experiment 4 took 0.35660925200000015s
Terminal States reached: 6
Steps   per terminal state: [951, 584, 339, 516, 859, 603]
Rewards per terminal state: [-447, -80, 165, -12, -355, -99]
```

Graphical representations of results

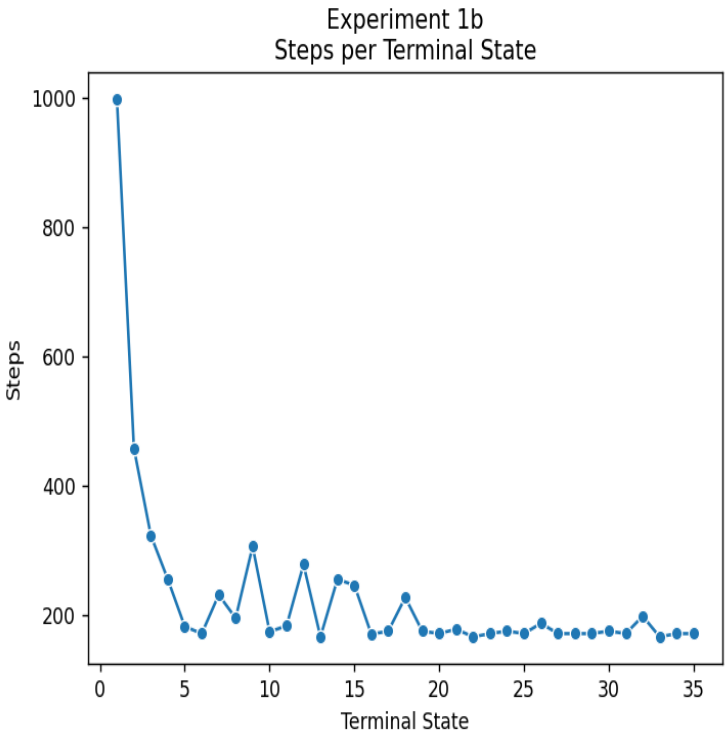
The following graphs depict the results for all experiments for seed=577 for single q-table and seed=326 for individual q-table for each agent. (Left side is single q-table and Right side is individual q-table)

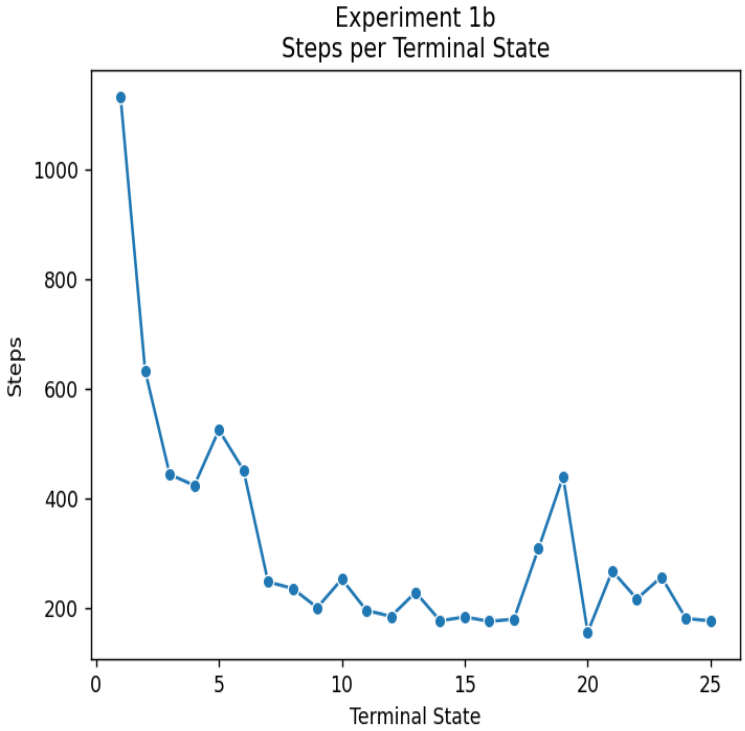
Experiment 1a:



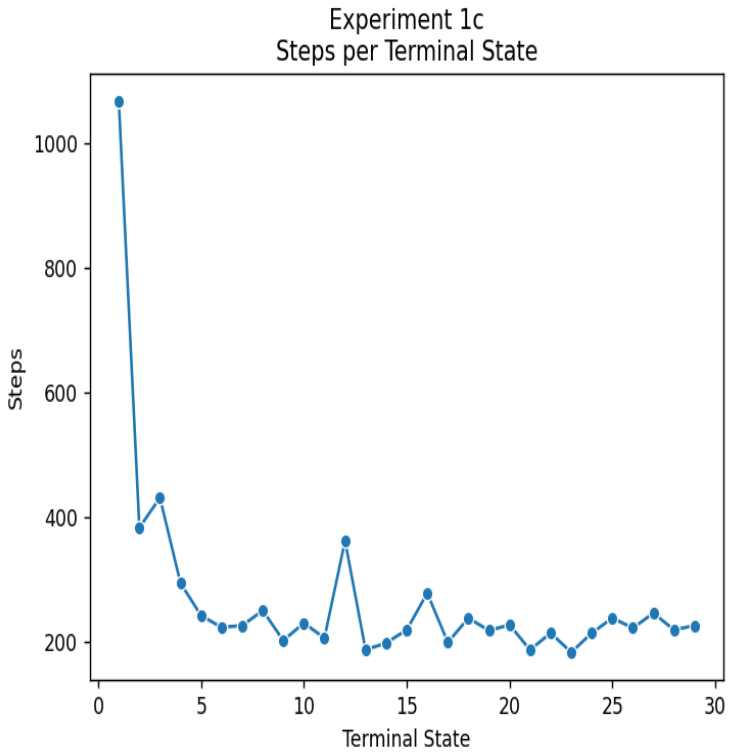


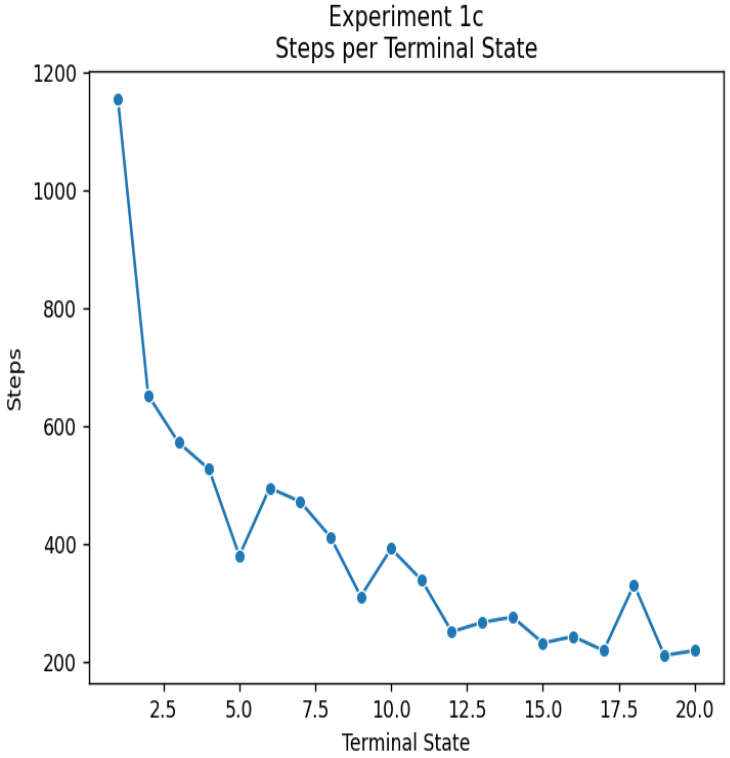
Experiment 1b:



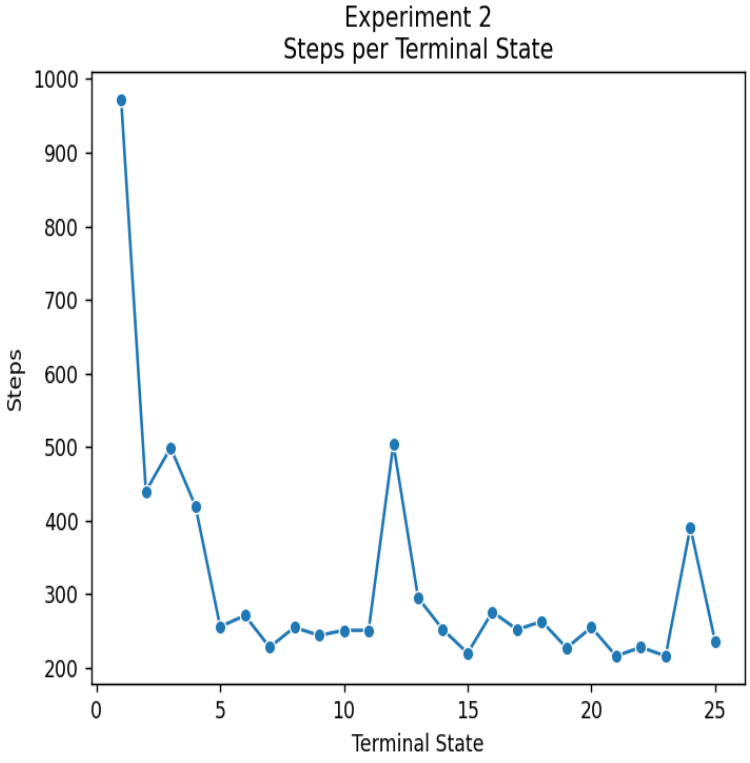


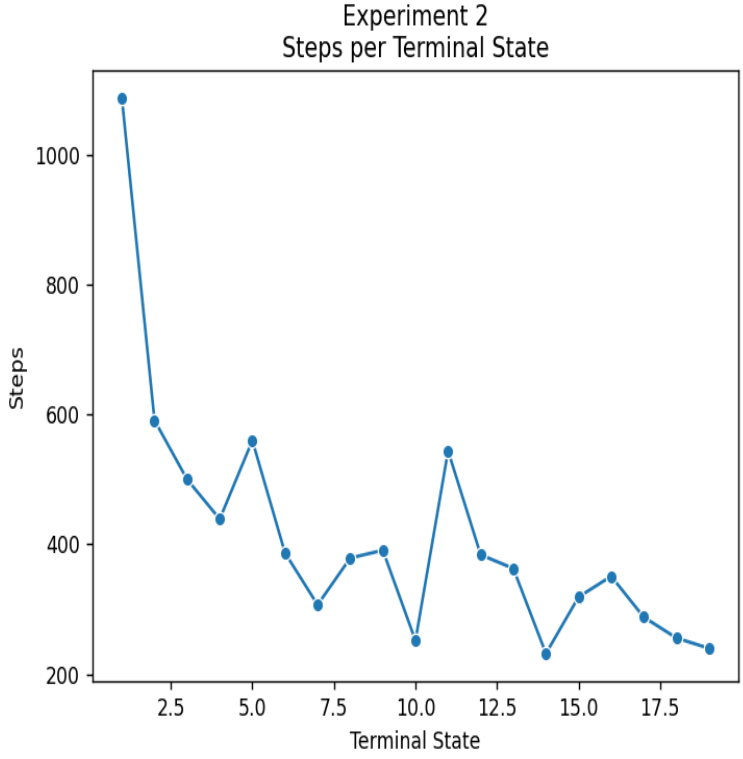
Experiment 1c:



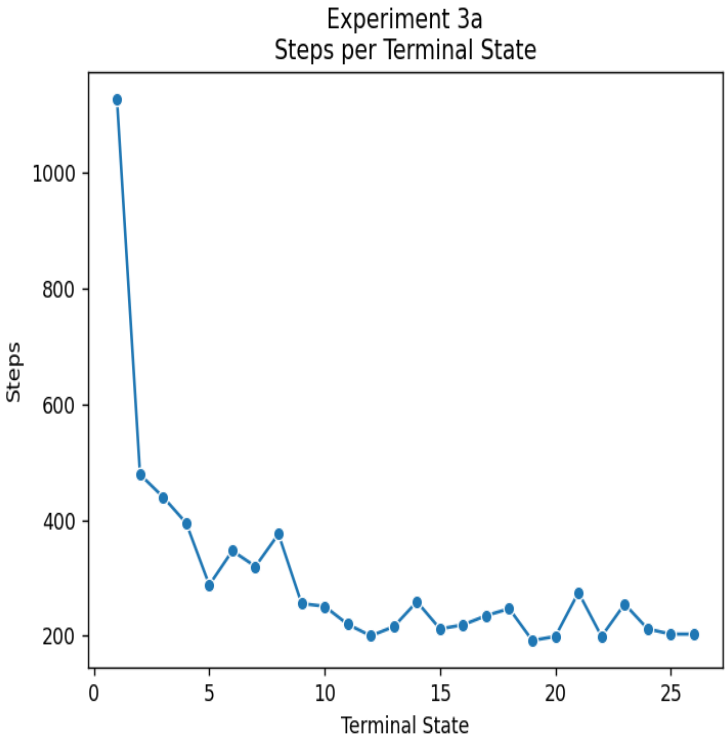


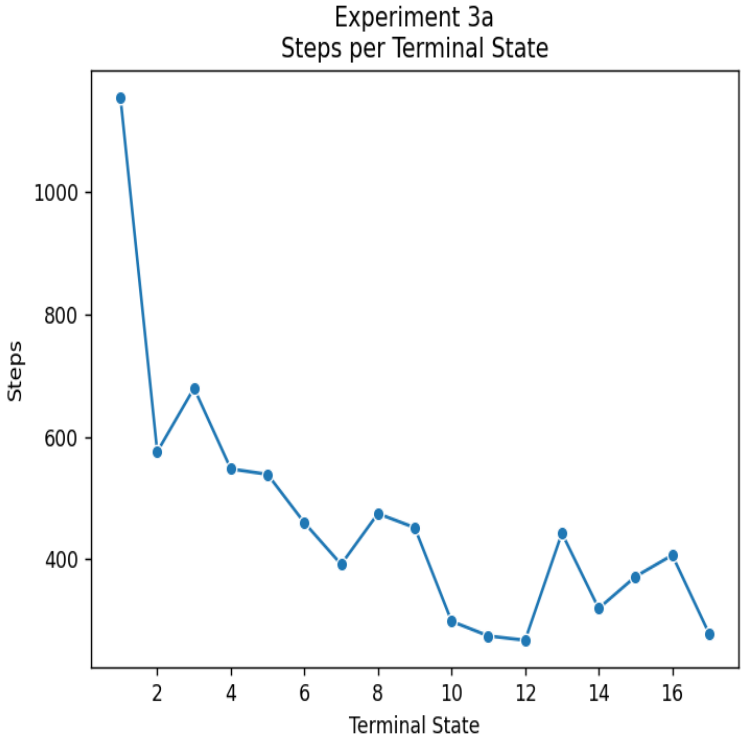
Experiment 2 using SARSA:



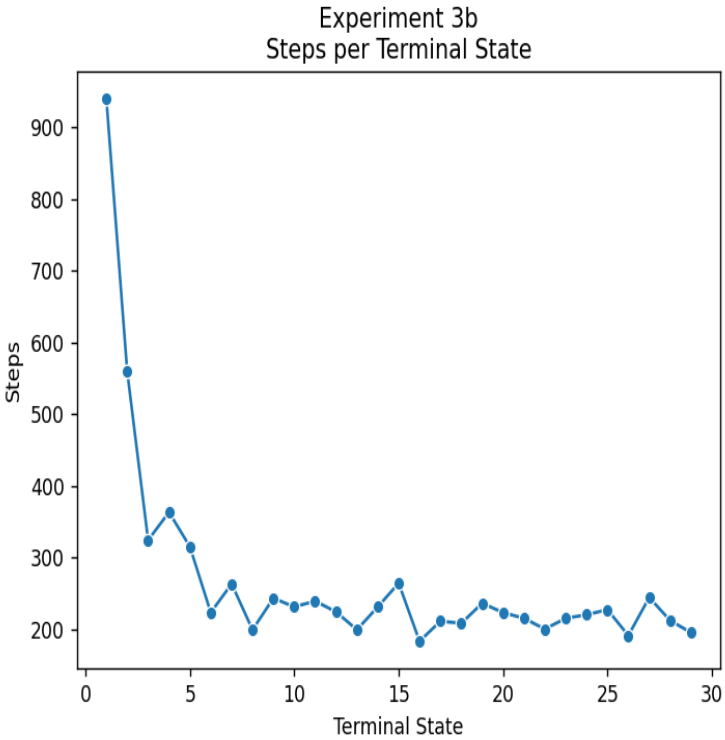


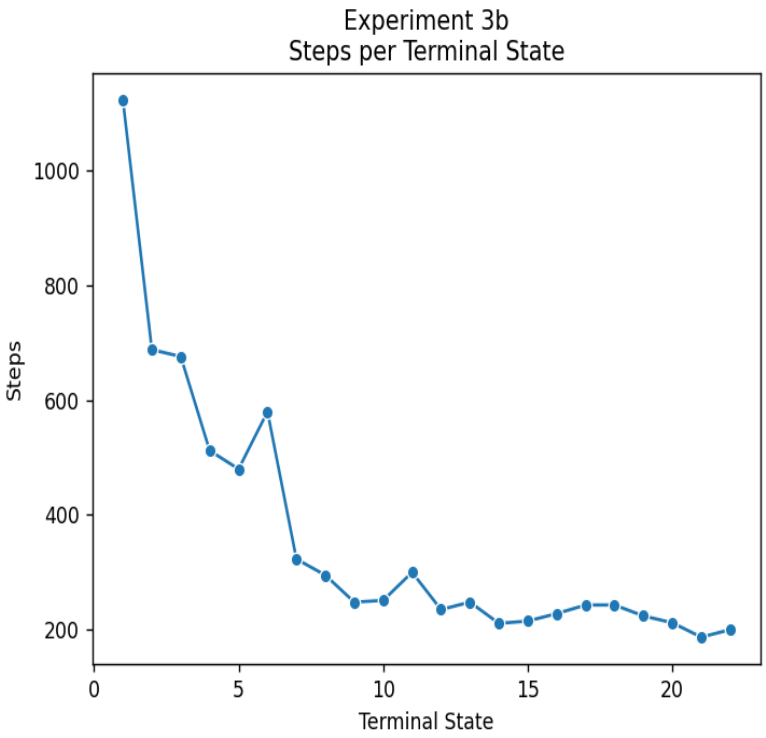
Experiment 3 with learning rate = 0.15:





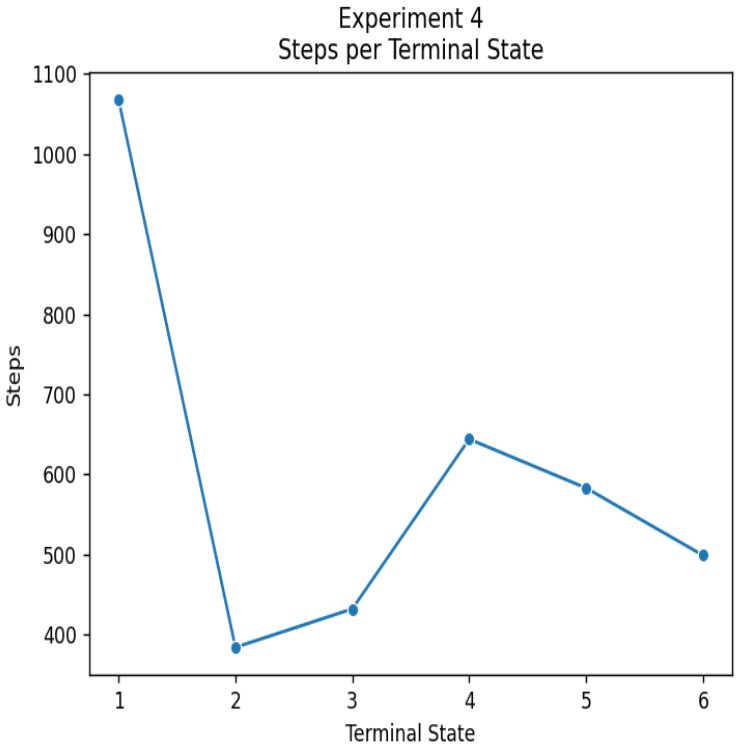
Experiment 3 with learning rate = 0.45:

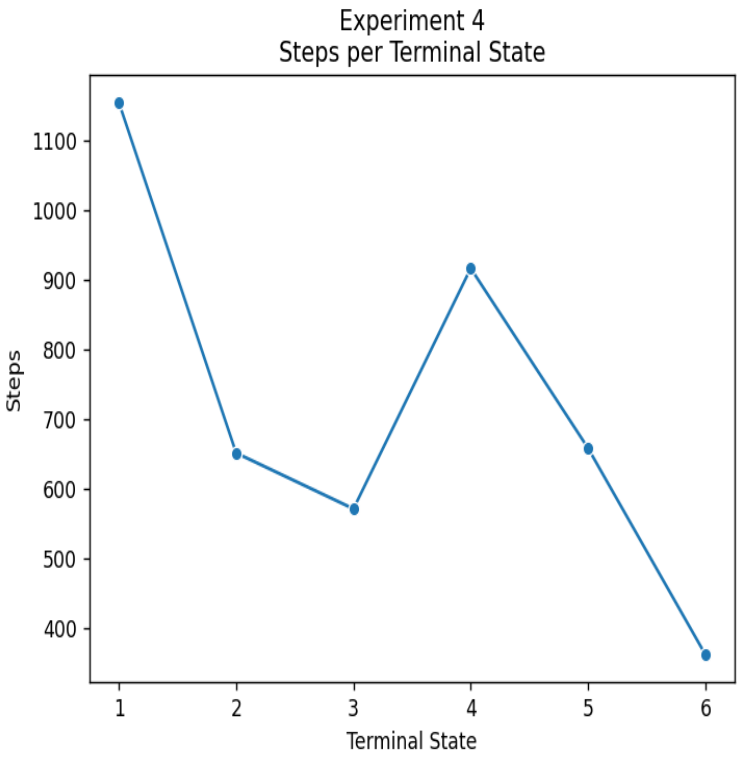




Experiment 4:

[changed pickup locations to (0,1), (3,4) at 3 terminal states, and ended experiment at 6 terminal states]





Interpretations

PExploit vs. PGreedy: Comparing experiments 1b and 1c

PExploit has an 80% chance of selecting the action with the highest Q-value, whereas PGreedy has a 100% chance of selecting the action with the highest Q-value. PExploit has a 20% chance of using a random action to continue to explore the state-action space in search of better paths whereas PGreedy more aggressively settles on a particular path earlier on. Theoretically speaking when the state-action space is already thoroughly explored PGreedy is more advantageous than PExploit as exploration is no longer necessary to finding an optimal path, but if the state-action space is not already thoroughly explored then PExploit will continue to improve as it explores and discovers more optimal paths while PGreedy would otherwise have already settled on a suboptimal path. Although PGreedy outperforms PExploit in terms of performance, due to a lack of exploration, when two agents block each other and are on the same path, they can change the q-value of the optimal path, resulting in an unending loop severely degrading performance. These characteristics of PExploit and PGreedy can be seen in a single q-table approach and two q-table approaches.

Q-learning vs. SARSA: Comparing experiments 1c and 2

Q-learning finds the optimal path by calculating the Q-value of each state. SARSA finds the optimal state by calculating the SARSA value of each action. SARSA will be able to choose a better course in the long term if

it learns from its actions. Experiment 1c yielded 29terminal states, while Experiment 2 yielded 25. In our experiment, the SARSA convergence issue is visible. In Q-learning, values converge quickly, so the number of steps per terminal state decreases significantly after a few terminal states, whereas in SARSA, it takes more steps for terminal steps, in other words, it takes more steps to converge q-values. Because of this convergence issue, SARSA appears to perform similarly to or slightly worse than Q-learning. The gap between q-learning and SARSA is minimal in independent reinforcement learning strategies and q-tables. They operate fairly similarly, which is likely due to the fact that in separate q-tables, the improper movements of one agent do not influence both agents, but in a single q-table, the wrong movements of one agent can affect both agents.

Learning rate: Understanding experiment 3

The learning rate is set to control the sight of learning, determining to what extent the newly acquired information will override the old information. It regulates the rate at which utilities are updated. We used learning rate = 0.15 and learning rate = 0.45 in experiment 3. Because the environment is unchanging, it turns out that a low learning rate works poorly. When compared to learning rate = 0.15, learning rate = 0.45 performs admirably. Separate reinforcement learning strategy and single reinforcement learning both show changes in performance related to learning rate.

Unlearning Paths: Understanding experiment 4

In experiment 4, we move the pick-up location to a new coordinate while maintaining the existing q-values. This update causes agents to repeat their steps to two previous pickup locations, resulting in a -1 reward instead of a +13. It takes multiple attempts for the agent to unlearn the old path and relearn the new path. In the graphs, it can be easily observed that after the third terminal state, the number of steps per terminal increases, and then progressively declines after relearning the new path as the agent learns new paths. This pattern can be seen in both reinforcement learning approaches.

Agent Coordination:

For understanding agent coordination, we measured the counts of instances where agents blocked each other. Agents block each other roughly 400 times on average, resulting in a loss of approximately 600 to 1000 total steps for each trial. Experiment 1b (PGREEDY) shows the most blocking, indicating that agents are likely working on the same pathways. Because agents follow the same path, they might get stuck in an endless circle of movement. Blocking is a concern in other policies as well, but due to the likelihood of exploration, such an extreme degradation of performance has not been observed. Blocking is significantly less in separate q-table approach indicating agents are working on different paths.

Hivemind vs Independent Agents

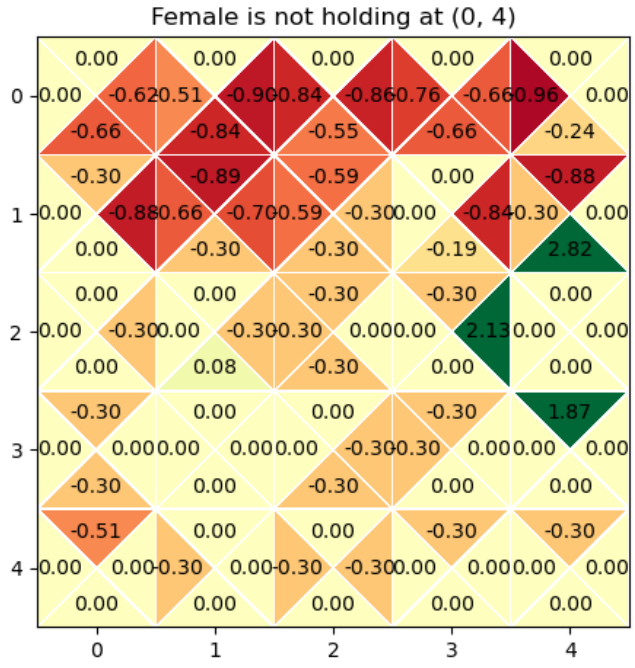
In the graphical illustration, the performance of a single Q-table and a separate Q-table for each agent can be easily observed. Agents clearly take longer to learn and converge their q-values, resulting in fewer terminal steps. The performance of experiments 1c(Q - learning) and 2(SARSA) was one notable observation. The performance of 1c and 2 grew increasingly similar. Experiment 3's performance is equal to that of a single Q-table method, with the exception of a lower terminal state. We can fairly state that having a single Q-table is superior for quicker learning after putting both approaches to the test. Furthermore, even with separate Q-tables, the endless cycle problem between two agents remains unresolved, hence there are many reasons to utilize a single reinforcement learning strategy.

Visualizing Attractive Paths with Heatmaps

Experiment 2 was run with seed=577 and various situations had heatmaps produced representing the q-table for both the female and male agents' holding statuses.

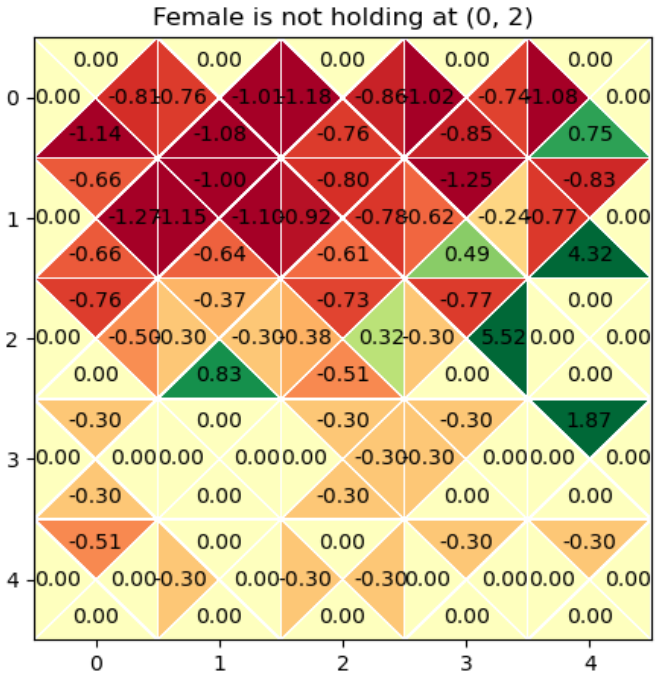
Situation a: When the first drop-off location is filled

First dropoff location filled was (0,4). Male agent is at (2, 3) and is also not holding so only the heatmap for the female agent is shown since they're the same.



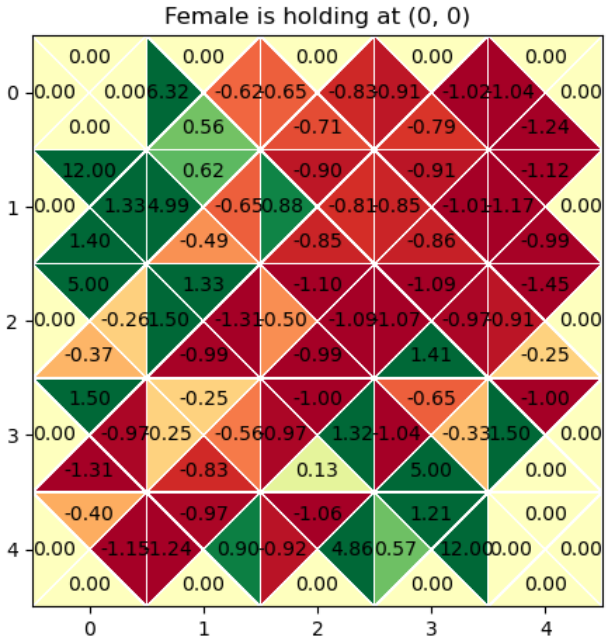
Situation b: When the first terminal state is reached

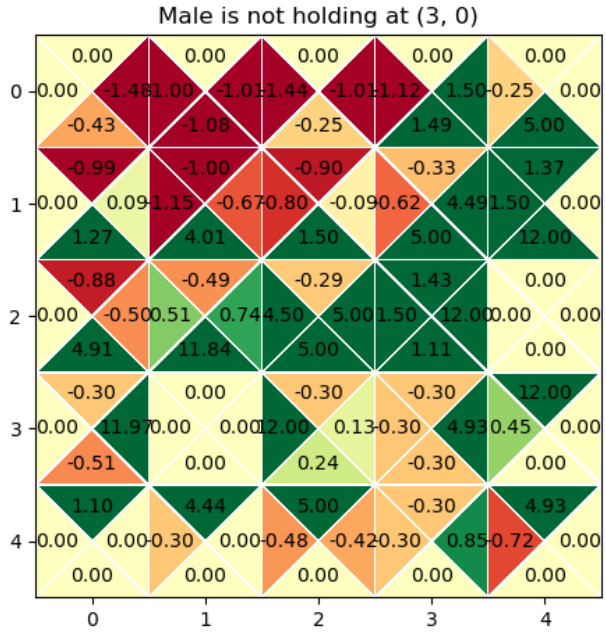
Male agent is at (4, 2) and is also not holding so only the heatmap for the male agent is shown since they're the same.



Situation c: The final q-table

Pickup index status = 3, dropoff index status = 7 (DISAMBIGUATE THIS INTO POSITIONS)





Visualization

The Unity Engine

Algorithmic visualization are performed using the Unity Engine and a message passage interface (MPI) system built to route execution of our models directly from C#'s dot.Net framework. A system process triggers a bash script which in turn executes the python driven models given an argument of preferred experiment. The instructional information is then fed back to the Unity system, determining the actions and movements of both the male (playerM) and female (playerF) characters. The option to generate a .txt file of player instructions per experiment is granted to the user.

Player Mechanics and Instructional Execution

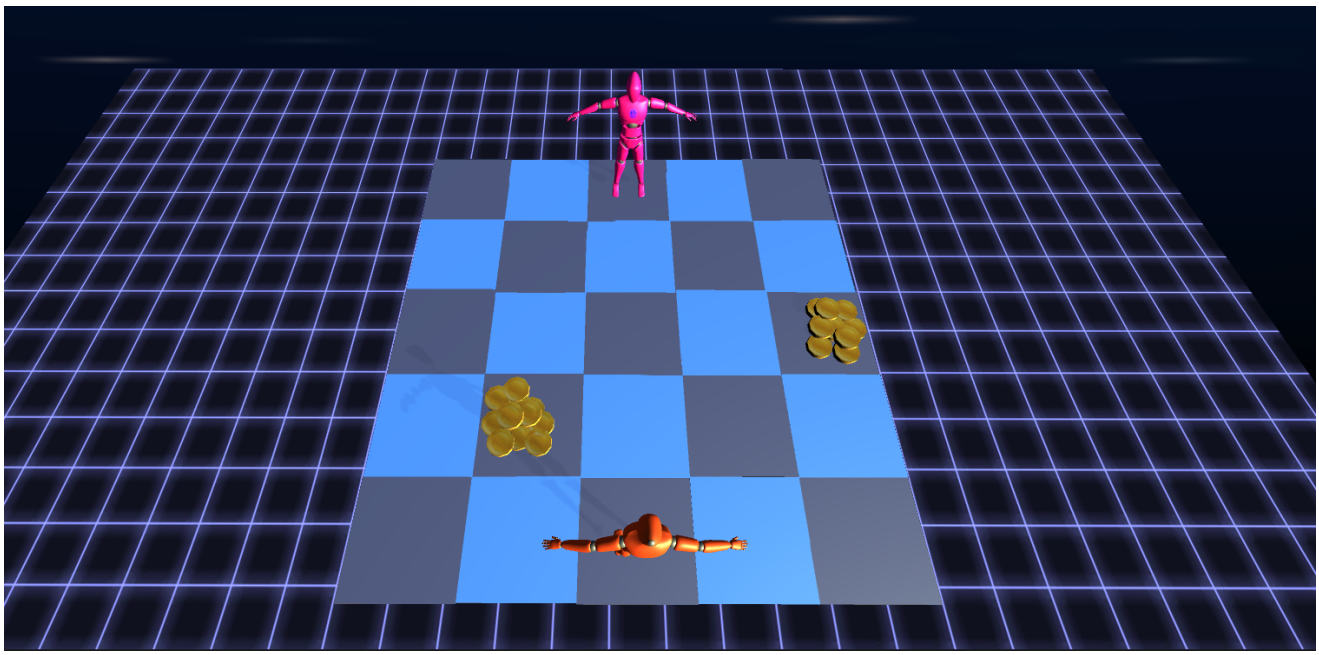
The players operate on delta time; a method of time calculating seconds passed per the previous frame. Because of the real time nature of delta time, developing a strong system of communication is crucial to precise instructional execution. A series of event listeners are implemented throughout both playerM and playerF's source code, with a central controller navigating player communication. This MPI system can be seen as the container for each player's brain. Rotational, translational, and all player mechanics are calculated dynamically in relation with both their own player bodies and the world space.

Aesthetics

The animations for playerM and playerF were built and tuned to allow for a clear and concise representation of each player's learning process. Design and color schema was also deeply considered to maintain an chic aesthetic to keep users engaged. Here we see our initial setup, with playerM (male) in orange and playerF (female) in fuschia. Surrounding them are the 20 tokens dispersed on each pick up location, with all game objects grounded on a 5x5 grid.

Future Visualization Development

As this project progressed, many doors were opened to the possibilities of how useful and innovative such a game-type visualization can be in the world of Reinforcement Learning. Given the constraints of time, we consider future work on this project to add further aesthetic effects, alternate constraints within the given models, and the implementation of additional learning based models



Conclusion

In this section, we will conclude a challenging but interesting project towards reinforcement learning. We first presented the problem statement and then presented our proposed approach to solve the 2 agent problem in a Pickup-Dropoff World in accordance with guidelines provided by the professor and teacher's aids. We implemented, analyzed, and interpreted a Multi-Agent Reinforcement Learning framework to make an efficient sequence of decisions for both the Q-Learning and SARSA algorithms while conducting 4 different experiments with different learning rates and policies of greedy, exploit, and random.

Extra credit approaches: 1) visualization of q-tables, 2) analysis of attractive paths, 3) methods to analyze agent coordination, 4) visually appealing visualization using Unity Engine, 5) Each agent uses his/her own reinforcement learning strategy and Q-Table and a single reinforcement learning strategy and Q-Table

References

[1]https://en.wikipedia.org/wiki/State%E2%80%93action%E2%80%93reward%E2%80%93state%E2%80%93action

[2]https://lilianweng.github.io/posts/2018-02-19-rl-overview/

[3]https://towardsdatascience.com/practical-reinforcement-learning-02-getting-started-with-q-learning-582f63e4acd9

[4]https://www.oreilly.com/library/view/hands-on-reinforcement-learning/9781788836524/20659243-cadb-46f0-b5c3-3acadd590d67.xhtml

[5]https://stats.stackexchange.com/questions/184657/what-is-the-difference-between-off-policy-and-on-policy-learning