COSC 4368 Group Project Report

1. **Introduction:**

This project introduced us to a subfield of artificial intelligence called reinforcement learning. The basis of reinforcement learning is to train agents to learn from their interaction with an environment through repeated trial and error. The agent takes certain actions in its environment and receives rewards based on those actions. The agent’s goal is to learn a policy from those actions to maximize the expected cumulative reward. In this experiment, we were introduced to Q-learning and SARSA, which are two popular reinforcement learning algorithms. We have designed and implemented a 3D 3x3x3 cube world in which two agents will traverse to try and accomplish transportation goals. The sole purpose of this experiment was to help us get a better understanding of reinforcement learning and to be able to apply reinforcement learning to a real problem.

The Q-learning and SARSA algorithms used are as follows:  
Q-Learning: Q(a,s) ß (1-a)\*Q(a,s) + a\*[R(s’,a,s)+ γ\*maxa’Q(a’,s’)]

SARSA: Q(a,s) ß Q(a,s) + α [ R(s) + γ\*Q(a’,s’) - Q(a,s) ]

1. **System/Software Design:**

This project was designed completely with Python using the libraries numpy and Vpython. Vpython was used to visualize the 3d world and the agents traversing it. So these libraries need to be installed. This project was designed in a modular fashion so we implemented a lot of the main elements of the experiment as modules. The modules include main, Qtable, RL, state, and world. Main file is the user controller assists us in running different experiments and add the frame rate for vpython environment. Qtable class contains two qtables with each assigned to respective agents in the PDWorld where the data structure is a matrix of positions values as a dictionary, with the inner dictionary to represent the various movement actions (“North”,”SOUTH”, “EAST”,”WEST”,”UP”, “DOWN”). Then the states of agent under first list as 4 possible conditions when in pickup state, (2^2), the inner list represents if agent is in dropoff state, (2^4) possible variations to check whether agent carries a block. State space is the position of the agent, along with pickup state or dropoff state values. Which gets passed to the reinforment learning.py. RL has all the main modules to perform the training for the agents in the PDWorld. World initiates the PDWorld, with grid cells, agents, pickup points, dropoff points, and other values to change later down the line. In consideration to the 3d environment this visualization is build upon vPython library which handles experimental programs to run efficiently, with maximum system performance and utilizing minimal power based on the rate at which the program is trained. Also, one thing to know is that we indexed our x,y, and z coordinated from 0. So we started our coordinates from 0 and went to 2. So for us (0,0,0) is the equivalent of (1,1,1) if it was indexed from 1. Initially, our female agent was in position (0,0,0) and our male agent was in position (2,1,2). (In experiment 4 our pickup locations change automatically so nothing needs to be manually set)

1. **Experiments and Results**

**1. Experiment 1:**

**Summary of Experiment 1:**

In this experiment, we used **α**=0.3 and γ=0.5 using the traditional Q-learning algorithm for 10000 steps. We will initially run PRANDOM for 500 steps then:

1. Run PRANDOM for 9500 more steps
2. Run PGREEDY for the remaining 9500 steps
3. Run PEXPLOIT for the remaining 9500 steps

**Pseudocode Experiment 1:**

**def experiment1(self, subExperiment):**

**for step in range(500):**

**agent, x, y,z, i = self.Turn(step)**

**self.POLICY(step, "PRANDOM")**

**if subExperiment == 'a':**

**self.experiment1\_a()**

**elif subExperiment == 'b':**

**self.experiment1\_b()**

**elif subExperiment == 'c':**

**self.experiment1\_c()**

**def experiment1\_a(self):**

**for step in range(500,9500):**

**agent, \_, \_,\_, \_ = self.Turn(step)**

**self.POLICY(step, "PRANDOM")**

**def experiment1\_b(self):**

**for step in range(500,9500):**

**agent, \_, \_,\_, holding = self.Turn(step)**

**self.POLICY(step, "PGREEDY")**

**def experiment1\_c(self):**

**for step in range(500,9500):**

**male, \_, \_,\_, \_ = self.Turn(step)**

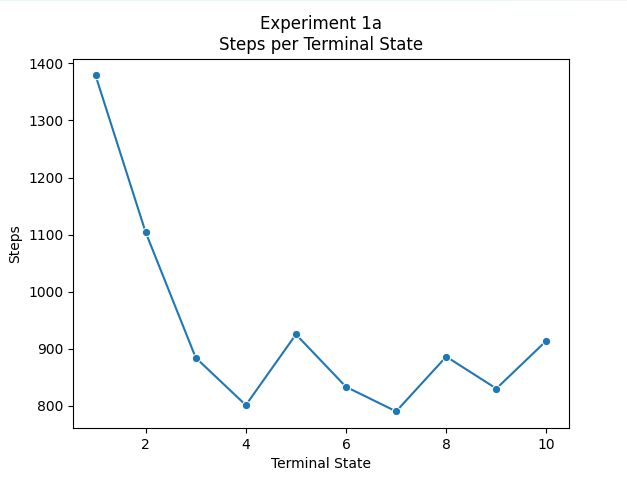
**self.POLICY(step, "PEXPLOIT")**

**Results of Experiment 1:**

The results we obtained from part A of the experiment are shown below in steps per terminal state graph and also the rewards per terminal state for both agents. As you can see in run 1 of our experiment for part A, we were able to reach the terminal state 10 times using the PRANDOM policy for the first 500 steps and then the additional 9500 steps. The number of steps taken to reach the terminal state significantly declined as our experiment kept running and after the second terminal state was reached. Our biggest drop in steps is in between the first terminal to the fourth terminal state, with a drop of 578 steps. In addition, our rewards per terminal state for both agents significantly increased as we reached more and more terminal states. Although, we did see some slight increases in between some terminal states. In run 2 of our experiment for PRANDOM, we produced completely different results. We reached 10 terminal states but there was an upward trend in the number of steps to reach each terminal state. But this is normal since this policy took completely random actions, so sometimes results will be better and sometimes results will be worse. In part B for run 1, we ran the PGREEDY policy for the remaining 9500 steps. And in this part, we were able to reach the terminal state 40 times, with the number of steps significantly decreasing to reach the next terminal state. We were able to reach the 40th terminal state in less than 200 steps. This is also coupled with the fact that our agents' combined rewards had a steady upward trend as each terminal state was reached. And only after reaching the second terminal state, we were able to obtain high positive rewards from there on out. The agent coordination from this experiment was very very optimal, this is proved by the fact that many terminal states were reached very quickly. And they were very close to the optimal subdivision of work. This makes sense because this policy was designed to choose the best q-value at every state the agent was in. For run 2 of part B, the agents were able to reach the terminal state 38 times and the results reached for this run were similar to the results reached in run 1 for the most part. In part C, for run 1, we ran the PGREEDY policy for the remaining 9500 steps, in this part we were able to reach the terminal state 31 times. so already this policy produced worse results than part B of the experiment. But this is normal since PEXPLOIT did have a more random element embedded in its policy type, which could’ve resulted in the agent going less optimal ways. Even though fewer terminal states were reached the number of steps needed to get to the next terminal state was always on a downward trend. In addition to this, the combined rewards for the agents also produced very high positive values, which were

very close to the PGREEDY rewards. Also, run 2 for this part of the experiment produced very similar results so anything that can be said about run 1 can also be said about run 2 for the most part. Both agents coordinated pretty well in both runs of the experiment and based on the results they were close to reaching the optimal subdivision of work. Although they weren’t as close as part B’s results.

**Run 1:**

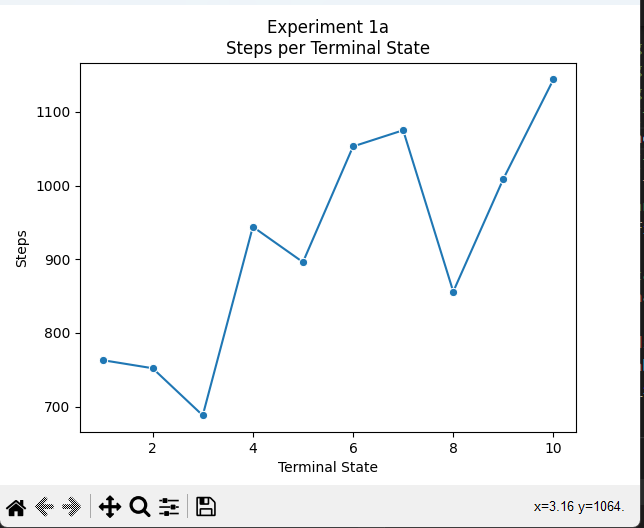
****

**Terminal States reached: 10**

**Steps per terminal state: [1379, 1104, 884, 801, 925, 833, 790, 886, 830, 914]**

**Rewards per terminal state: [-952, -648, -431, -332, -463, -357, -331, -421, -353, -453]**

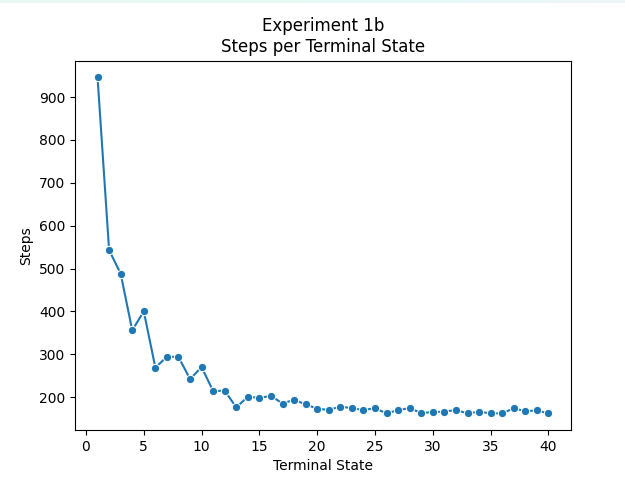
**Run 2:**



**Terminal States reached: 10**

**Steps per terminal state: [763, 752, 688, 944, 896, 1053, 1075, 856, 1009, 1144]**

**Rewards per terminal state: [-284, -279, -204, -478, -424, -594, -617, -380, -542, -679]**



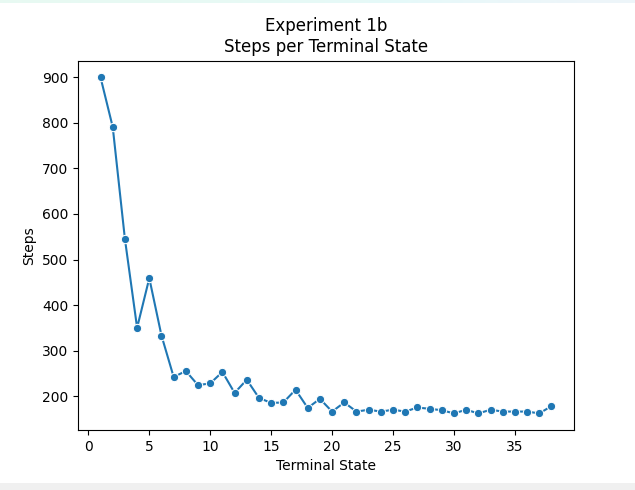
**Run 1:**

**Terminal States reached: 40**

**Steps per terminal state: [946, 543, 488, 356, 400, 269, 294, 294, 242, 270, 214, 215, 176, 201, 198, 203, 185, 194, 183, 173, 170, 178, 174, 170, 174, 162, 170, 174, 163, 165, 166, 170, 162, 166, 162, 162, 174, 167, 169, 162]**

**Rewards per terminal state: [-496, -47, 13, 157, 103, 253, 229, 226, 285, 252, 317, 314, 357, 333, 332, 327, 349, 335, 348, 357, 364, 357, 360, 365, 357, 373, 364, 360, 371, 370, 369, 362, 372, 369, 373, 373, 359, 365, 365, 373]**

**Run 2:**

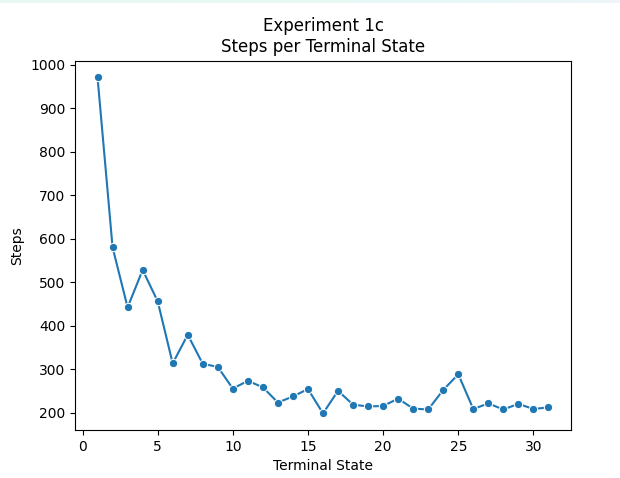
****

**Terminal States reached: 38**

**Steps per terminal state: [900, 791, 545, 350, 459, 333, 242, 255, 224, 228, 253, 207, 236, 196, 185, 186, 214, 174, 194, 166, 186, 166, 170, 166, 170, 166, 175, 172, 169, 162, 170, 162, 170, 166, 166, 166, 162, 178]**

**Rewards per terminal state: [-453, -319, -55, 177, 45, 182, 283, 271, 302, 303, 270, 323, 293, 336, 348, 348, 315, 358, 340, 368, 345, 368, 365, 368, 364, 366, 356, 361, 365, 373, 365, 373, 363, 368, 369, 369, 373, 356]**

**Run 1:**

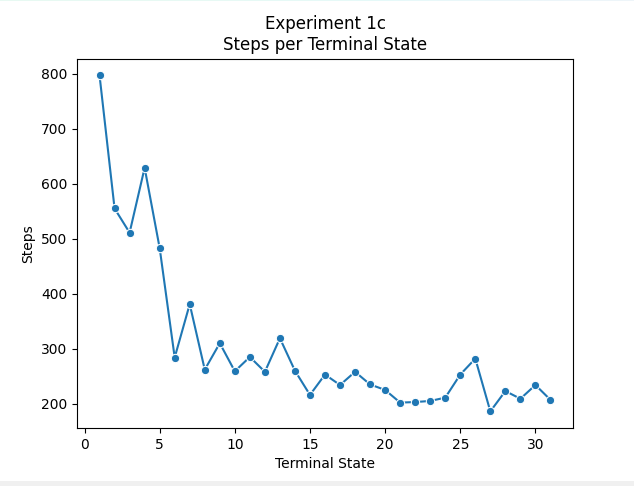
****

**Terminal States reached: 31**

**Steps per terminal state: [971, 580, 442, 528, 456, 313, 379, 312, 305, 255, 273, 258, 223, 237, 254, 198, 250, 218, 214, 215, 232, 209, 207, 252, 288, 208, 221, 207, 220, 208, 212]**

**Rewards per terminal state: [-528, -87, 65, -31, 52, 205, 134, 203, 215, 275, 251, 268, 305, 288, 274, 335, 271, 311, 316, 312, 292, 321, 324, 272, 229, 322, 301, 322, 305, 322, 318]**

**Run 2:**

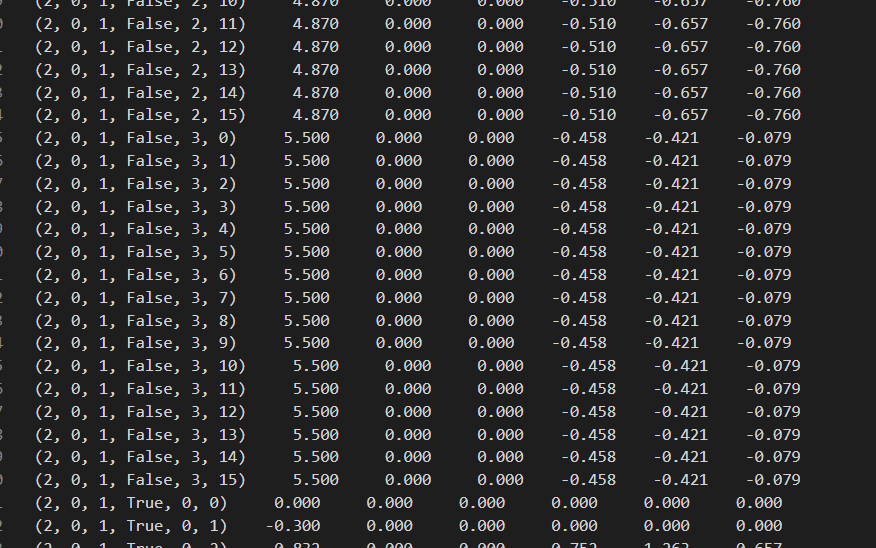
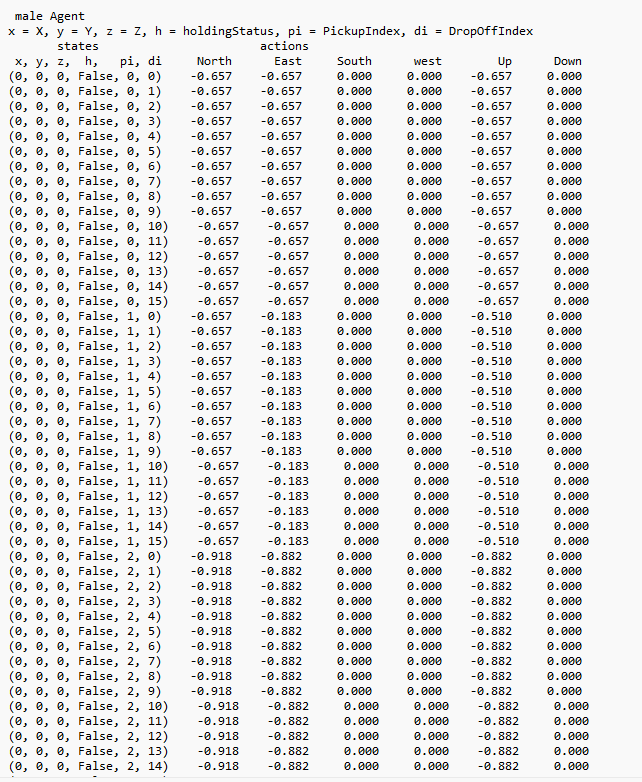
****

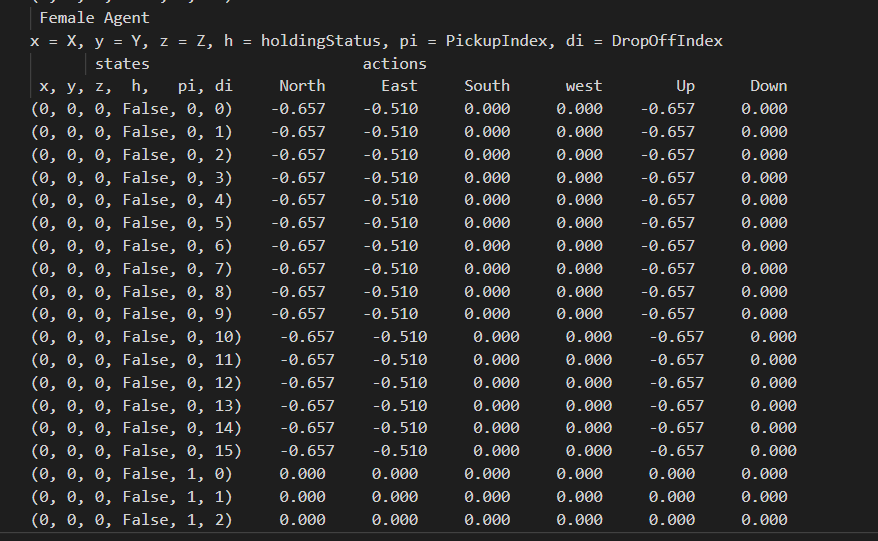
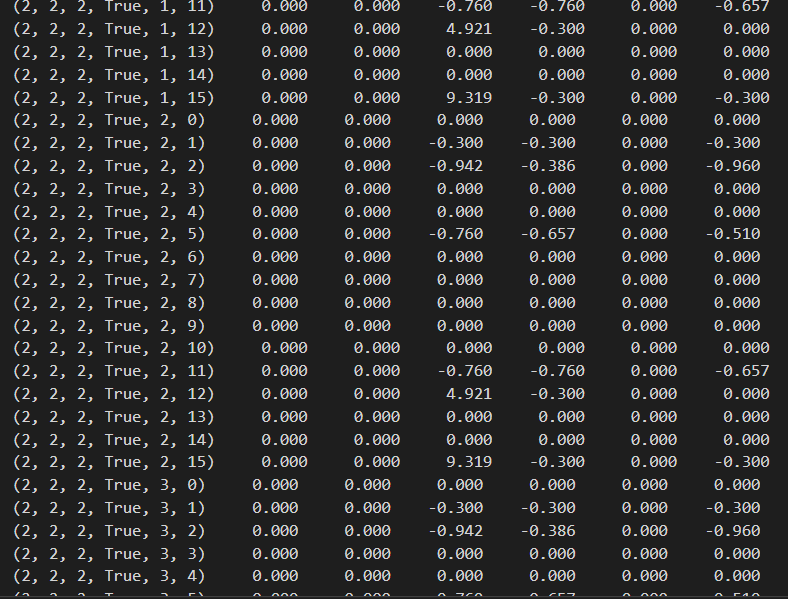
**Terminal States reached: 31**

**Steps per terminal state: [797, 555, 511, 629, 483, 284, 381, 262, 310, 259, 285, 258, 319, 260, 216, 253, 234, 258, 235, 225, 202, 203, 205, 211, 253, 282, 186, 223, 209, 234, 207]**

**Rewards per terminal state: [-330, -66, -20, -136, 13, 235, 130, 260, 211, 272, 237, 268, 205, 267, 312, 273, 294, 267, 289, 302, 331, 327, 325, 319, 270, 242, 347, 310, 321, 290, 321]**

**Final Qtables for the male agent and female agent (whole q-table was too big to fit, these are just some rows):**

****

** **

**2. Experiment 2:**

**Summary of Experiment 2:**

This experiment will be the exact same as experiment 1c, but will now run the SARSA Q-learning variation. Since we needed to compare the

**Pseudocode Experiment 2:**

**def experiment2(self):**

**print("Experiment 2 ", end="")**

**for step in range(500):**

**male, \_, \_,\_, \_ = self.Turn(step)**

**for step in range(500,9500):**

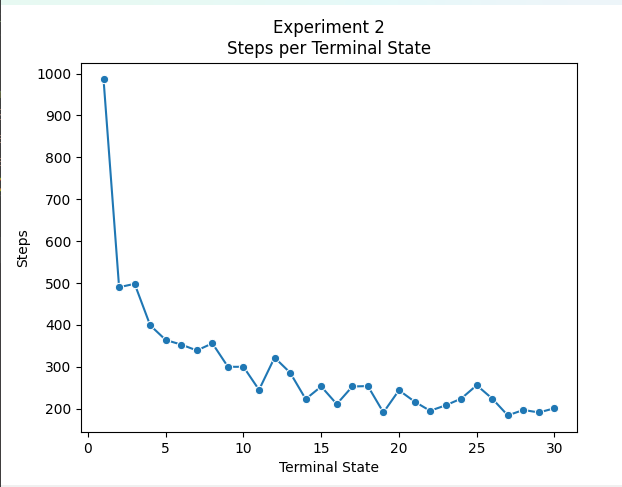
**male, \_, \_,\_, \_ = self.Turn(step)**

**self.POLICY(step, "PEXPLOIT")**

**Results of Experiment 2:**

In experiment 2, we ran the PEXPLOIT policy but this time we used the SARSA Q-learning variation for 9500 steps. Since we wanted to compare the two Q-learning variations we also decided to run experiment 1c again twice for our personal testing purposes, denoted underneath Q-learning, before running the SARSA algorithm and the results produced are very similar to the results produced in experiment 1c. Now after running the SARSA algorithm twice, we saw that it was only able to reach the terminal state 10 times for both and the combined reward for the agents were all very low negative values. It did not look like the agents were getting better at reaching their terminal state as the agents progressed. The number of steps taken to reach the next terminal state just seemed random, which is possible since the policy used was PEXPLOIT. Now comparing the two experiments, Q-learning without a doubt did better at reaching its terminal states compared to SARSA. In Q-learning the number of steps taken to reach the next terminal step was on a consistent downward slope. This is also coupled with the fact that the rewards for the agents in Q-learning were always significantly higher than SARSA. Based on our results, SARSA did not do as well a job in agent coordination and work subdivision as Q-learning did.

**Q-Learning:**

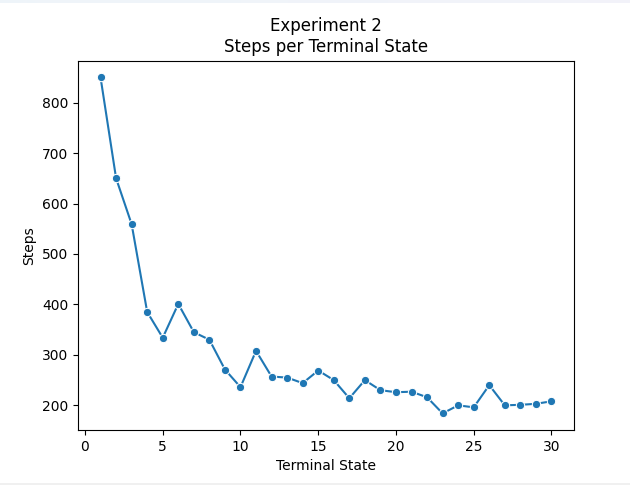
**Run 1:  
**

**Terminal States reached: 30**

**Steps per terminal state: [986, 490, 498, 399, 364, 353, 339, 356, 300, 300, 245, 322, 286, 223, 253, 211, 253, 254, 191, 244, 217, 195, 208, 224, 256, 224, 184, 197, 191, 201]**

**Rewards per terminal state: [-536, 11, 1, 104, 154, 163, 184, 159, 224, 216, 276, 191, 240, 307, 276, 316, 271, 272, 341, 278, 316, 337, 322, 304, 262, 306, 350, 333, 346, 332]**

**Run 2:**

****

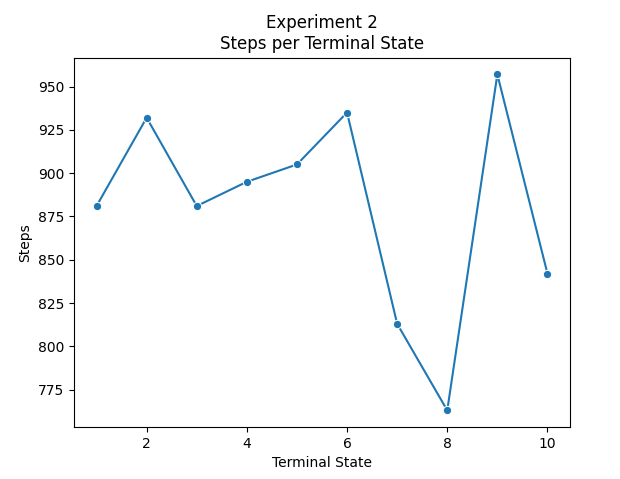
**Terminal States reached: 30**

**Steps per terminal state: [850, 650, 559, 385, 334, 401, 345, 330, 271, 236, 308, 257, 255, 244, 269, 250, 214, 250, 230, 226, 227, 216, 184, 200, 196, 240, 200, 201, 203, 208]**

**Rewards per terminal state: [-393, -164, -82, 135, 184, 112, 171, 192, 247, 289, 215, 268, 275, 284, 253, 282, 318, 282, 301, 306, 303, 322, 351, 332, 335, 280, 335, 328, 333, 325]**

**SARSA:**

**Run 1:**

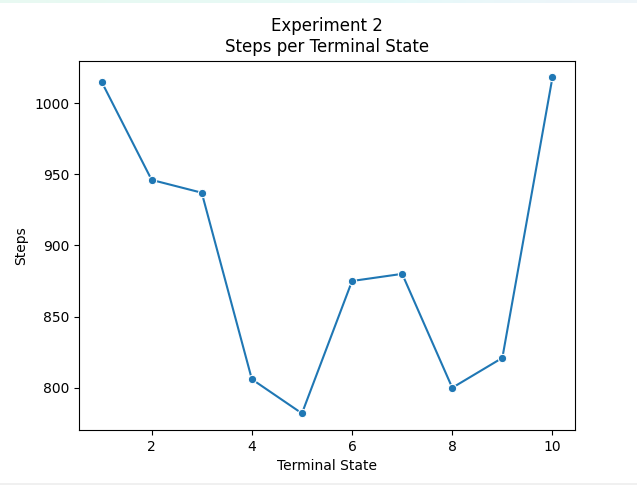
****

**Terminal States reached: 10**

**Steps per terminal state: [881, 932, 881, 895, 905, 935, 813, 763, 957, 842]**

**Rewards per terminal state: [-433, -473, -422, -423, -455, -476, -342, -284, -508, -374]**

**Run 2:**

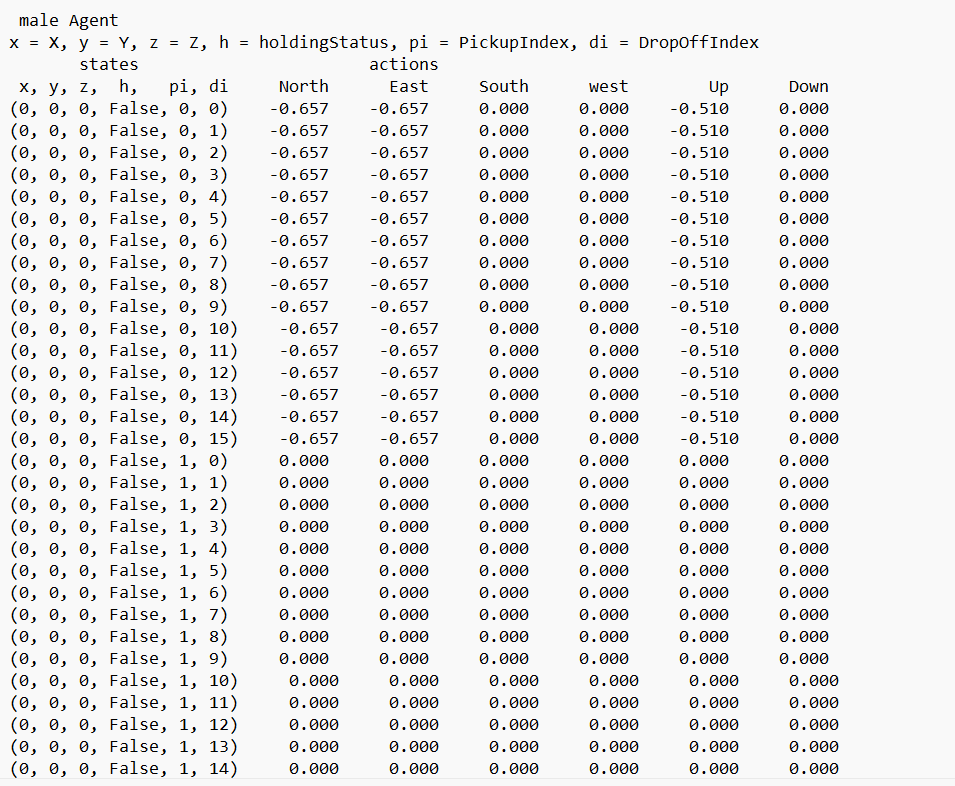
****

**Terminal States reached: 10**

**Steps per terminal state: [1015, 946, 937, 806, 782, 875, 880, 800, 821, 1018]**

**Rewards per terminal state: [-574, -483, -478, -334, -312, -410, -411, -337, -359, -571]**

**Q-Table for experiment 2:**

****

**3. Experiment 3:**

**Summary of Experiment 3:**

In this experiment, we could choose to rerun experiment 1c or experiment 2. We decided to rerun experiment 1c with the new learning rates.

**Pseudocode Experiment 3:**

**def experiment3(self):**

**for step in range(500):**

**male, \_, \_,\_, \_ = self.Turn(step)**

**self.POLICY(step, "PRANDOM")**

**for step in range(500,9500):**

**male, \_, \_,\_, \_ = self.Turn(step)**

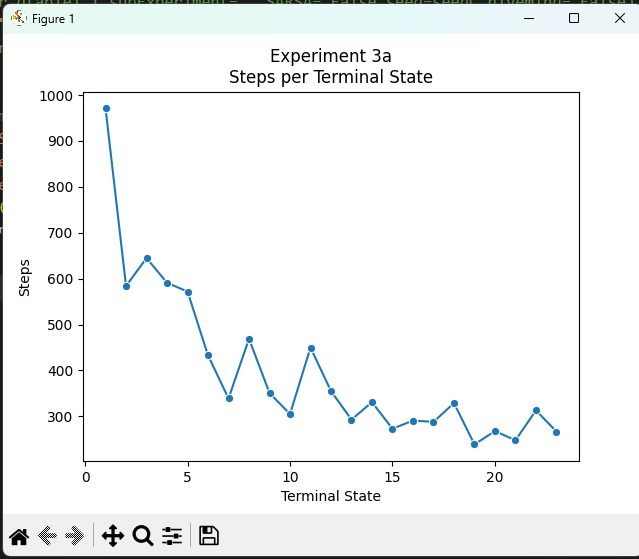
**self.POLICY(step, "PEXPLOIT")**

**Results of Experiment 3:**

In experiment 3 we decided to run experiment 1c again with the different learning rates. The graphs below show the results of each experiment and the learning rate that was used. So after analyzing the results, when a = .1, run 1 and run 2 reached the terminal state 24 times. When a = .5, run 1 achieved 32 terminal states, and run 2 achieved 34 terminal states. When a = .3, run 1 achieved 31 terminal states, and run 2 achieved 29 terminal states. And in all three experiments, the number of steps taken to reach each terminal state also decreased as a new terminal state was reached. So going by this trend, we believe that increasing the learning rate improved the performance of the agents reaching more terminal states and also slightly decreased the number of steps to reach each terminal state. When a=.5 in run 1 it reached its last terminal state in 200 steps compared to when a=.3 in run 1 it reached its last terminal state in 207 steps. The difference is not by much but we do see a slight trend of increasing learning rate with increasing experiment performance.

Agent coordination also slightly improved as well when learning rates increased.

**Run 1 and a=0.1:**

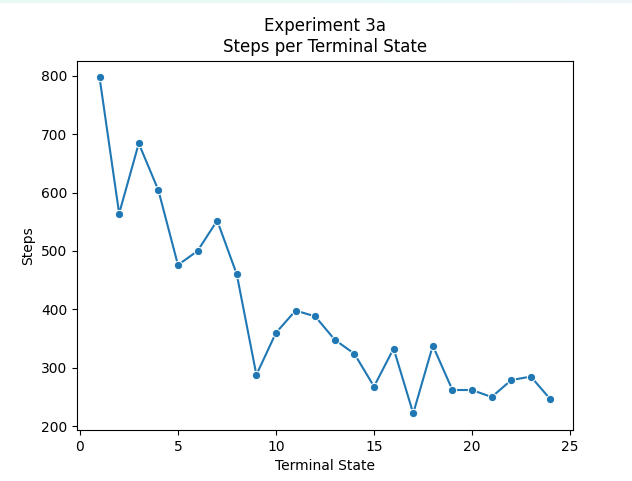
****

**Terminal States reached: 24**

**Steps per terminal state: [1003, 577, 618, 359, 452, 625, 426, 339, 345, 323, 301, 431, 264, 329, 326, 290, 338, 279, 289, 295, 241, 255, 252, 261]**

**Rewards per terminal state: [-553, -89, -141, 154, 52, -134, 82, 176, 171, 189, 219, 72, 263, 189, 195, 236, 176, 248, 236, 220, 279, 273, 278, 265]**

**Run 2 and a=0.1:**

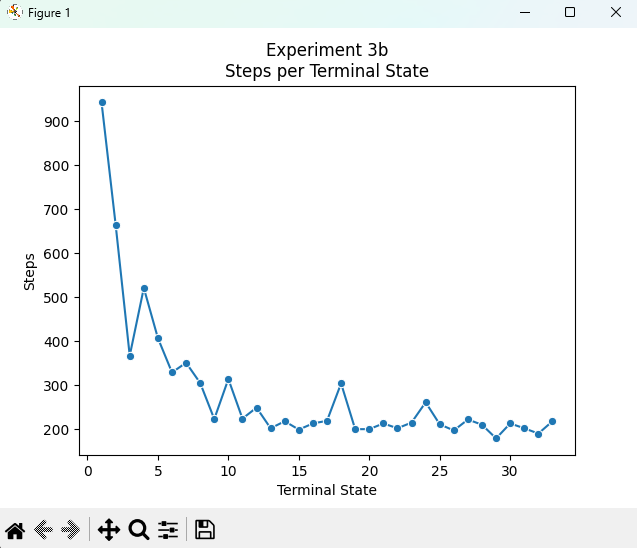
****

**Terminal States reached: 24**

**Steps per terminal state: [797, 564, 684, 604, 476, 500, 552, 460, 288, 360, 398, 388, 348, 324, 268, 333, 222, 338, 262, 262, 250, 279, 285, 246]**

**Rewards per terminal state: [-330, -66, -208, -116, 25, 3, -54, 54, 238, 151, 110, 129, 170, 190, 254, 188, 307, 180, 268, 264, 273, 248, 233, 281]**

**Run 1 and a=.5:**

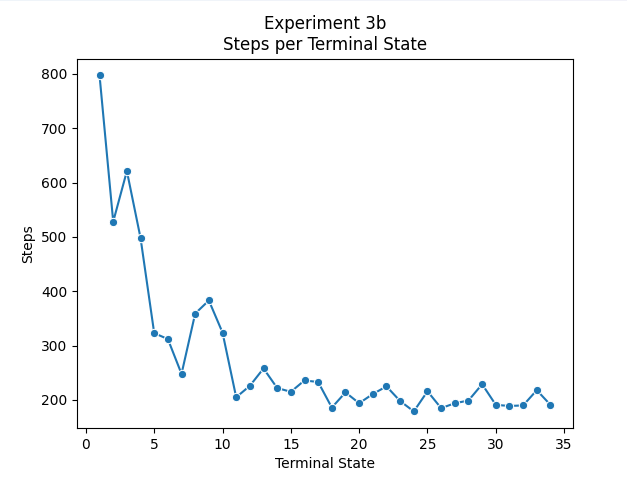
****

**Terminal States reached: 32**

**Steps per terminal state: [958, 581, 409, 546, 363, 345, 366, 331, 276, 268, 277, 290, 259, 228, 221, 243, 193, 267, 201, 191, 189, 266, 235, 196, 205, 258, 178, 203, 193, 227, 196, 200]**

**Rewards per terminal state: [-495, -105, 98, -56, 150, 171, 150, 195, 253, 256, 245, 235, 267, 297, 306, 286, 337, 250, 333, 344, 346, 264, 300, 340, 333, 271, 357, 332, 342, 304, 340, 332]**

**Run 2 and a=.5:**

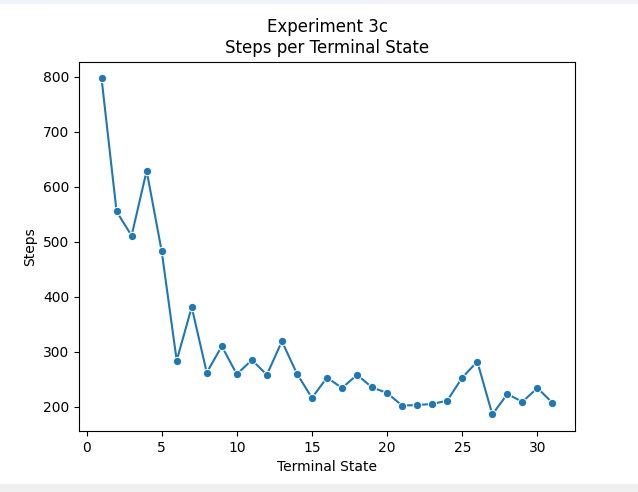
****

**Terminal States reached: 34**

**Steps per terminal state: [797, 528, 621, 498, 323, 312, 248, 359, 383, 324, 205, 226, 258, 222, 215, 236, 233, 186, 214, 194, 211, 225, 198, 179, 216, 185, 194, 199, 229, 191, 189, 190, 218, 191]**

**Rewards per terminal state: [-330, -46, -139, 6, 188, 201, 271, 149, 129, 196, 321, 303, 266, 308, 312, 289, 291, 349, 313, 333, 318, 302, 332, 353, 316, 346, 332, 329, 298, 343, 346, 346, 312, 346]**

**Run 1 and a=.3:**

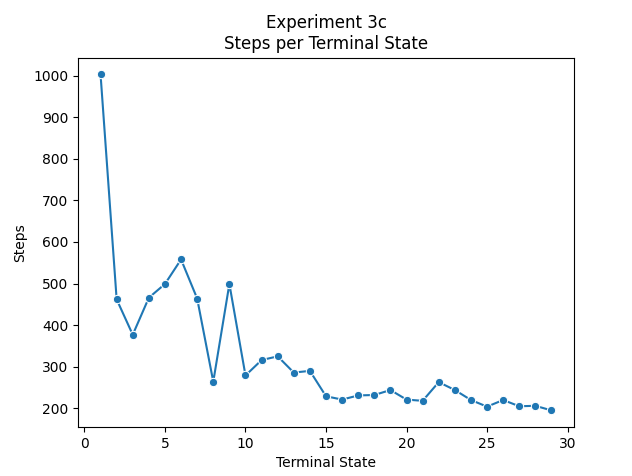
****

**Terminal States reached: 31**

**Steps per terminal state: [797, 555, 511, 629, 483, 284, 381, 262, 310, 259, 285, 258, 319, 260, 216, 253, 234, 258, 235, 225, 202, 203, 205, 211, 253, 282, 186, 223, 209, 234, 207]**

**Rewards per terminal state: [-330, -66, -20, -136, 13, 235, 130, 260, 211, 272, 237, 268, 205, 267, 312, 273, 294, 267, 289, 302, 331, 327, 325, 319, 270, 242, 347, 310, 321, 290, 321]**

**Run 2 and a=.3:**

****

**Terminal States reached: 29**

**Steps per terminal state: [1003, 462, 377, 466, 499, 558, 464, 264, 499, 279, 316, 325, 286, 290, 229, 221, 231, 232, 244, 221, 218, 263, 244, 220, 204, 220, 205, 206, 195]**

**Rewards per terminal state: [-553, 47, 134, 39, -4, -73, 42, 267, -1, 244, 204, 189, 242, 233, 307, 312, 303, 303, 288, 306, 315, 267, 286, 313, 329, 310, 326, 325, 341]**

**4. Experiment 4:**

**Summary of Experiment 4:**

In this experiment, we will use **α**=0.1 and γ=0.5 just like in the experiments before, and we can use either Q-learning or SARSA. We decided to use \_\_\_\_\_\_. Then we will run PRANDOM for the first 500 steps, then we run PEXPLOIT. But after a terminal state is reached for the third time, the pick-up locations will be changed to (2,3,3) and (1,3,1). In our case (1,2,2) and (0,2,0) because we indexed from 0. The drop-off locations and Q-table will not be changed. We will also continue to run PEXPLOIT until the agent reaches a terminal state for the sixth time.

**Pseudocode Experiment 4:**

**def experiment4(self):**

**for step in range(500):**

**male, \_, \_,\_, \_ = self.Turn(step)**

**self.POLICY(step, "PRANDOM")**

**for step in count()**

**male, \_, \_,\_, \_ = self.Turn(step)**

**self.POLICY(step, "PEXPLOIT")**

**if self.terminalStatesReached == 3:**

**break**

**self.qTable[(1, 1, 0)].pop('Pickup')**

**self.qTable[(2, 2, 1)].pop('Pickup')**

**self.pickUpCells = [(1, 2, 2), (0, 2, 0)]**

**self.resetWorld()**

**self.qTable[(1, 2, 2)]['Pickup'] = [[0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]**

**self.qTable[(0, 2, 0)]['Pickup'] = [[0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]**

**self.qTable\_[(1, 1, 0)].pop('Pickup')**

**self.qTable\_[(2, 2, 1)].pop('Pickup')**

**self.qTable\_[(1, 2, 2)]['Pickup'] = [[0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]**

**self.qTable\_[(0, 2, 0)]['Pickup'] = [[0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]**

**for step in count():**

**self.POLICY(step, "PEXPLOIT")**

**if self.terminalStatesReached == 6:**

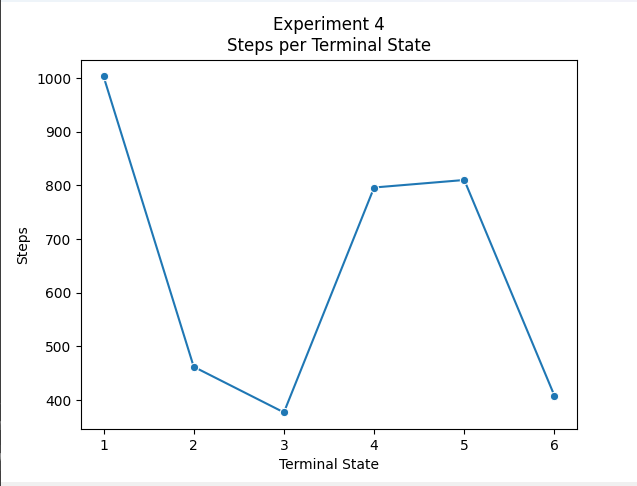
**Break**

**Results of Experiment 4:**

For this experiment, we decided to run our Q-learning algorithm since this algorithm has been performing the best on all our other experiments thus far. So based on the results from this experiment, for both runs, we can conclude that our learning strategy was able to adapt to change very well. We see that in both runs, once the third terminal state was reached our steps were in the process of converging and gradually declining. But once our pickup locations changed, we see an increase in steps to reach the fourth terminal state. But afterward, we see that the steps were able to converge and decline once the sixth terminal state was reached. In both runs both agents were able to converge very well into their terminal states. And both runs show that our agents were able to learn new paths and unlearn old paths in an efficient manner.

**Q-learning:**

**Run 1:**

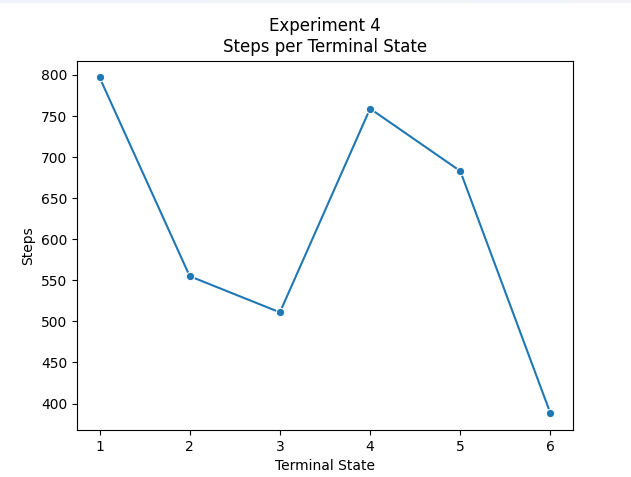
****

**Terminal States reached: 6**

**Steps per terminal state: [1003, 462, 377, 796, 810, 408]**

**Rewards per terminal state: [-553, 47, 134, -310, -343, 104]**

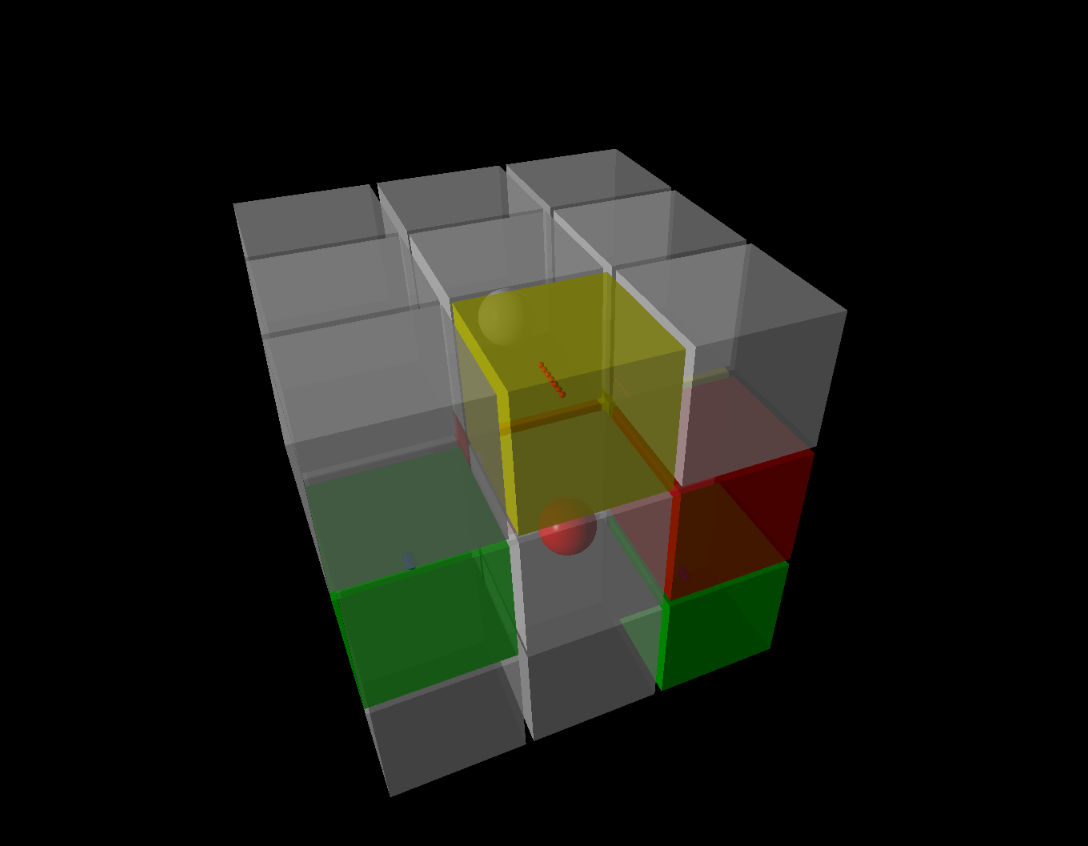
**Run 2:**

****

**Terminal States reached: 6**

**Steps per terminal state: [797, 555, 511, 759, 683, 388]**

**Rewards per terminal state: [-330, -66, -20, -278, -187, 131]**

****

**Analysis of All Experiments:**

After completing all experiments and seeing all of their results, we came to the conclusion that the best results came from run 1 and run 2 of experiment 1B. Run 1 was able to reach the terminal state 40 times and run 2 was able to reach it 38 times. Each time the number of steps it took to reach the next terminal state almost always declined and the combined rewards for both agents at each terminal state were very high positive values. This indicates that our agents were splitting up the work optimally and were coordinating very well with each other. Part B of experiment one also incorporated the PGREEDY policy so this outcome was somewhat expected since the agent will always pick the direction with the best possible q-value. In contrast to the other policies which incorporated randomness.So after analyzing all of our q-tables, we wanted to see some attractive paths our agents consistently took. But, our q-tables were way too large to try and identify attractive paths in each of them. But after browsing through our q-tables we did see certain operator q-values increase as the agents progressed through the environment, so our q-values did increase and there definitely were attractive paths, but to go through each of them would take way too long. Most notably we saw that from position (0,0,0) the most popular actions to take were east or up.

We believe that as our system was able to solve more PD world problems our system did gradually get better throughout all of our experiments. The steps needed to converge to our terminal states did gradually decline as we conducted our experiments. This showcases the fact that our agents were learning well and able to coordinate well with each other. Throughout all of our experiments and all of our runs for each experiment, the results produced were extremely similar. Almost all experiments followed the same pattern when converging to their terminal states. But the results for some runs of the same experiment were significantly different from each other. Most notably, the experiments that used either the PRANDOM or PEXPLOIT. The runs of the experiments that produced the most significantly different results were experiment 1a and experiment 2. In experiment 1a, the number of steps it took to reach each terminal state on run 1 pretty much decreased the more terminal states the agents reached. But in run 2, it was the complete opposite and steps increased as more terminal states were reached. But, this is acceptable because it was using the PRANDOM policy, so this is to be expected. In experiment 2, we used the SARSA variant with the PEXPLOIT policy. In this experiment in both runs it seemed like after each terminal state was reached the next amount steps to reach the next terminal state just seemed random. So both of our graphs for each run are significantly different. And again, this is okay because PEXPLOIT does have some randomness to it, so results sometimes will not be consistent. For our fourth experiment, on both runs, the graphs outputted were very similar. In both runs the graph followed the same pattern. Steps were decreasing as it was reaching the third terminal state and then spiked once we switched the pickup locations and then decreased again once it reached the sixth terminal state. So our agents were able to adapt to change very well and reached their last terminal states in a very low amount of steps.