Report

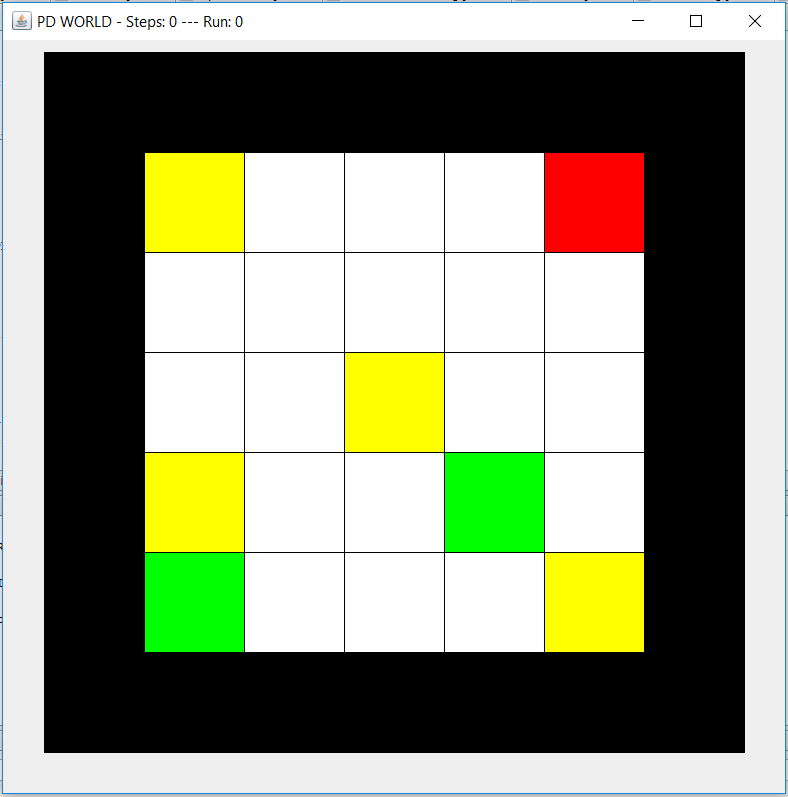
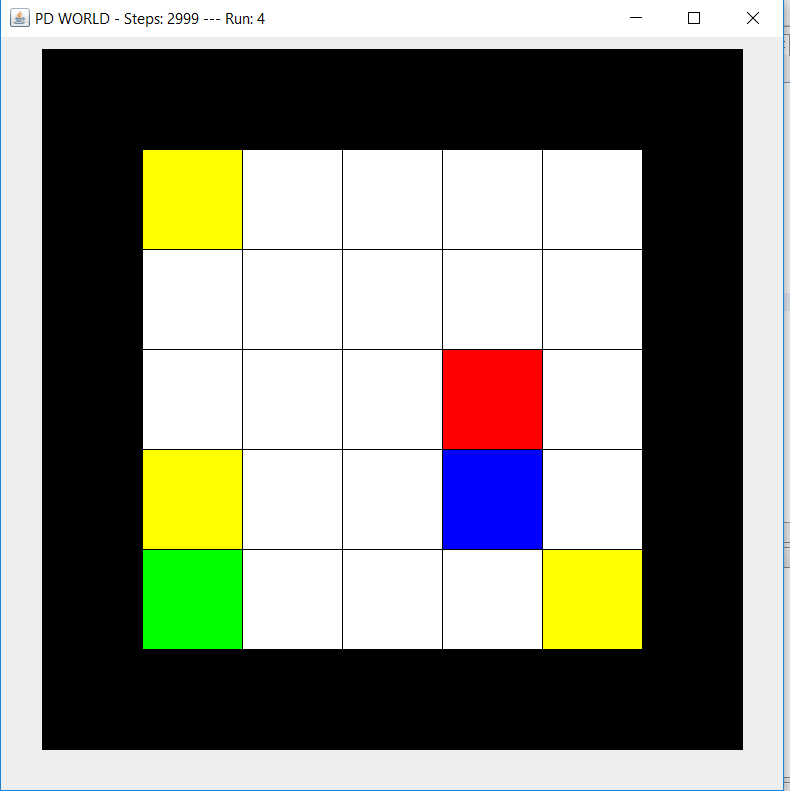
Learning Paths from Feedback Using Q-Learning and SARSA

In this project, I implemented Q learning and SARSA using Java and JFrame on NetBeans IDE. JFrame was used for visualization of moving the agent. The only difference between Q-learning and SARSA is just a learning function. Q-learning updates Q value with current state and next state with nest Q-value which may not be operated for next step since it may choose a different applicable operator randomly with probability 0.15 in exploit policy, but SARSA uses the q-value of the actual chosen action and not the q-value of the best action.

Q-Learning: Q (a, s) = (1-alpha) \* Q (a, s) + alpha\*[Reward (s’, a, s) + gamma \* max’Q (a’, s’)]

SARSA: Q (a, s) = Q (a, s) + alpha [ R(s) + gamma\*Q (a’, s’) – Q (a, s)]

At first, I used JFrame to implement PD World. I used 7 \* 7 array to store all positions since 1 to 5 would be easier to use for the coordinates compare to 0 to 5. I changed the colors and repainted the grids to make the agent looks like it was moving. I used red to denote the agent, yellow for flowers, green for drop off place that would turn to blue if it was full, white for other movable places, and black for the wall which was inapplicable. The program would be terminated when the number of moves had reached the maximum steps. The agent would restart from (1, 5) when all drop off places were full which was terminal state.

After that random policy had been implemented. I started implementing reinforcement learning methods and other functions. There are three policies that have been used for choosing an operator.

* **PRANDOM**: If pickup and drop-off is applicable, choose this operator; otherwise, choose an applicable operator randomly.
* **PEPLOIT**: If pickup and drop-off 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 q-value) with probability 0.85 and choose a different applicable operator randomly with probability 0.15.
* **PGREEDY**: If pickup and drop-off 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 q-value).

For the Q-Table, first column is the state, second column is up, third column is down, fourth column is left, and fifth column is right.

**The experiments are as follows:**

1. **In Experiment 1**, I used α=0.3, and run the Q-learning algorithm for 3000 steps with policy PRANDOM. It has completed 4 runs in 3000 steps:

run#: 1 --- steps: 626 --- bank accounts: -210

run#: 2 --- steps: 850 --- bank accounts: -434

run#: 3 --- steps: 892 --- bank accounts: -476

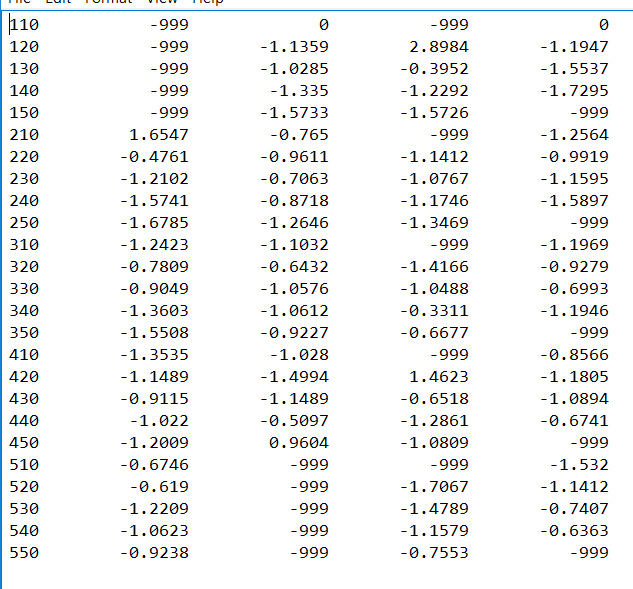
state = current\_x \* 100 + current\_y \*10 + carryingBlock.

Example: 110 means the state of non-carryingBlock on (1, 1).

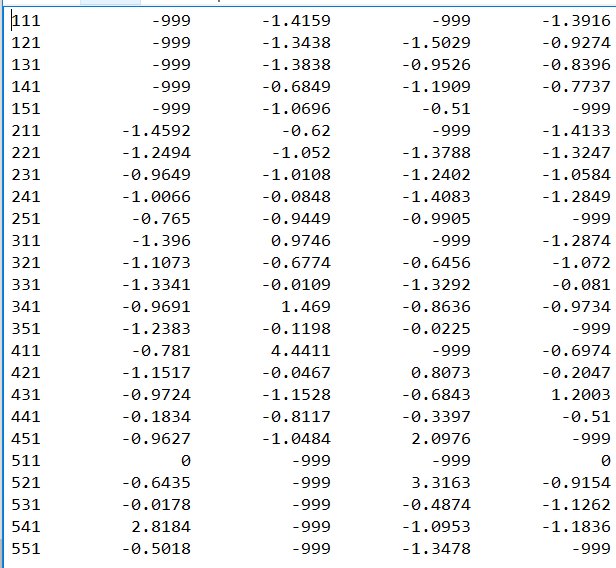
-999 on Q-Table means not applicable.

The Q-Tables I obtained on 1500th step:

No carrying block Q-Table:



carrying block Q-Table:



The Q-Tables I obtained in the end (I used green color for some Q-values that lead the agent to good position):

No carrying block Q-Table:

110 -999 0 -999 0.0357

120 -999 -1.2677 1.678 -0.6282

130 -999 -0.729 0.7297 -1.5332

140 -999 -1.3594 -1.0271 -1.7662

150 -999 -1.6494 -1.5593 -999

210 3.0909 -1.0411 -999 -1.3011

220 0.8368 -0.5542 -1.5654 -0.9267

230 -0.6938 0.9564 -1.3031 -1.3273

240 -1.55 -0.9649 -0.5878 -1.6706

250 -1.7801 -1.3083 -1.3863 -999

310 -1.4903 1.0899 -999 -0.5872

320 -1.2619 0.8336 -0.8051 -0.2256

330 -0.5862 -1.0929 -0.8925 -0.8585

340 -1.4221 -1.1163 -0.7508 -1.3153

350 -1.6664 -0.4247 -0.895 -999

410 -0.9452 -1.5232 -999 0.7247

420 -0.3677 -1.6624 3.2122 -0.8179

430 2.0967 -1.5313 0.9038 -1.1177

440 -1.0002 -1.3003 -0.835 -0.1434

450 -1.3084 2.5156 -1.1717 -999

510 0.1809 -999 -999 -1.7087

520 1.2489 -999 -1.7992 -1.5252

530 -0.7937 -999 -1.7253 -1.3792

540 -1.1384 -999 -1.5778 -1.1819

550 -0.6493 -999 -1.1417 -999

Carrying block Q-Table:

111 -999 -1.1947 -999 -1.7979

121 -999 -1.6046 -1.6059 -1.6797

131 -999 -1.4251 -1.7724 -1.4035

141 -999 -0.8518 -1.6687 -1.6488

151 -999 -1.4074 -1.3811 -999

211 -1.6055 -0.3644 -999 -1.6126

221 -1.7957 -1.2174 -1.2032 -1.4223

231 -1.698 -0.9663 -1.612 -0.863

241 -1.4129 0.2791 -1.42 -1.4123

251 -1.69 -0.8565 -0.8608 -999

311 -1.1959 1.2869 -999 -1.2296

321 -1.6057 -0.4373 -0.3951 -0.867

331 -1.4256 -0.2978 -1.2458 0.1032

341 -0.8306 2.9837 -0.9364 -0.8766

351 -1.4056 0.2374 -0.042 -999

411 -0.4755 4.5947 -999 -0.478

421 -1.208 0.918 1.1861 -0.2835

431 -0.9275 -0.7863 -0.4367 1.2641

441 -0.1448 -1.0065 -0.3482 -0.1504

451 -0.8511 -1.6126 3.6898 -999

511 0 -999 -999 0

521 -0.4911 -999 3.9817 -0.8855

531 -0.3108 -999 0.4838 -1.4205

541 4.404 -999 -0.8178 -1.712

551 0.3223 -999 -1.4556 -999

The agent moved randomly, so it was not able to know which path was better. However, the Q-Table had been updated well by its function on some positions even though I used PRANDOM. For example, when the agent did not carry a block on (4, 5) or (5, 4), it would go down or right to the position (5, 5) that had flowers if we used PEXPLOIT or PGREEDY policy by this Q-Table. This was because that Q-Table got the reward by that step whatever it was randomly moved or not. The Q-value was always updated in correct way. Therefore, the agent still could learn with PRANDOM and get some good Q-values those were near to drop-off location if it had a block and to pick up location if it had no block.

**For Greedy Policy**: it has completed 10 runs in 3000 steps:

run#: 1 --- steps: 380 --- bank accounts: 36

run#: 2 --- steps: 422 --- bank accounts: -6

run#: 3 --- steps: 210 --- bank accounts: 206

run#: 4 --- steps: 126 --- bank accounts: 290

run#: 5 --- steps: 176 --- bank accounts: 240

run#: 6 --- steps: 454 --- bank accounts: -38

run#: 7 --- steps: 246 --- bank accounts: 170

run#: 8 --- steps: 226 --- bank accounts: 190

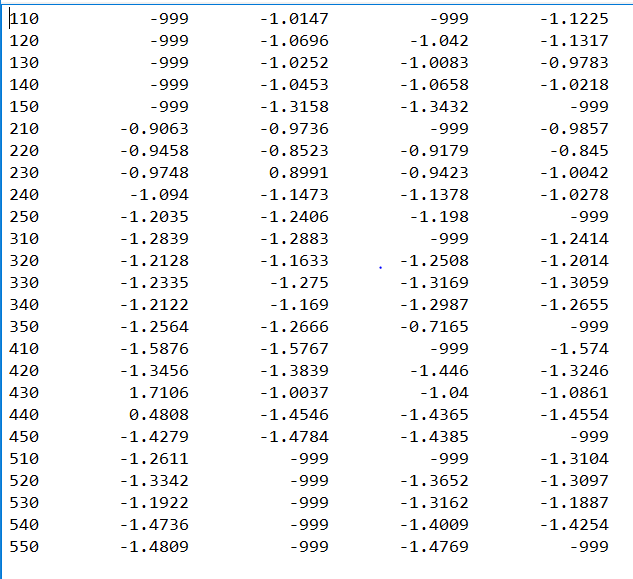
run#: 9 --- steps: 222 --- bank accounts: 194

run#: 10 --- steps: 328 --- bank accounts: 88

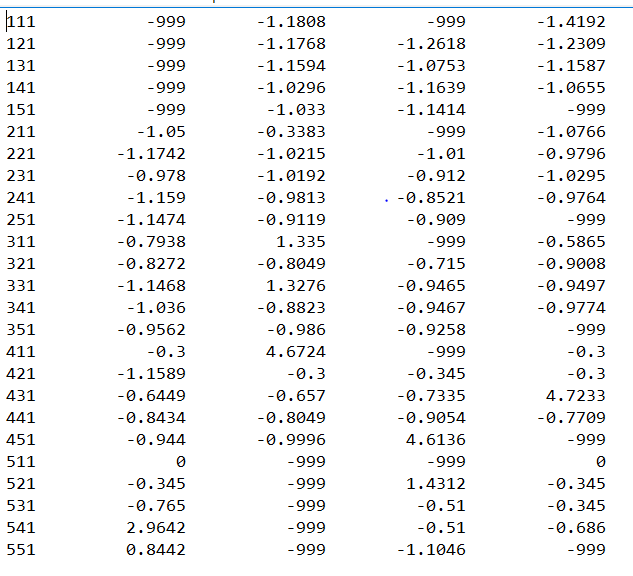
This graph shows that the agent has been learned with greedy policy since the number steps has been decreased and the balance of bank accounts has been increased. Sometimes, the agent got worse path than previous run since the Q-Table was created by previous experience that may not be appropriate for current run.

The Q-Tables I obtained on 1500th step:

No carrying block Q-Table:



Carrying block Q-Table:



The Q-Tables I obtained on 3000th step (I used green color for some Q-values that lead the agent to good position):

No carrying block Q-Table:

110 -999 -1.8028 -999 -1.8015

120 -999 -1.7889 -1.7911 -1.7766

130 -999 -1.7151 -1.7474 -1.7554

140 -999 -1.7791 -1.747 -1.7493

150 -999 -1.7971 -1.7819 -999

210 -1.7418 -1.6811 -999 -1.7285

220 -1.7168 -1.7426 -1.7222 -1.7174

230 -1.7114 -1.7638 -1.7348 -1.7393

240 -1.7452 -1.7432 -1.7601 -1.7548

250 -1.7791 -1.7832 -1.7464 -999

310 -1.7681 -1.7868 -999 -1.8006

320 -1.7837 -1.8002 -1.7982 -1.7744

330 -1.7915 -1.7937 -1.8058 -1.7948

340 -1.7766 -1.776 -1.8052 -1.8019

350 -1.7719 -1.4972 -1.7536 -999

410 -1.8507 -1.8443 -999 -1.8472

420 -1.8002 -1.8081 -1.8178 -1.7897

430 -1.7873 -1.7806 -1.7496 -1.7764

440 -0.6929 -1.7865 -1.7849 -1.762

450 -1.7529 1.5468 -1.7578 -999

510 -1.4038 -999 -999 -1.7484

520 -1.7926 -999 -1.7617 -1.7967

530 -1.7499 -999 -1.7762 -1.7624

540 -1.7604 -999 -1.7448 0.1956

550 -1.7391 -999 -1.7105 -999

Carrying block Q-Table:

111 -999 -1.1814 -999 -1.6426

121 -999 -1.4778 -1.5559 -1.5225

131 -999 -1.384 -1.4308 -1.414

141 -999 -1.3451 -1.2788 -1.314

151 -999 -1.2859 -1.2902 -999

211 -1.3832 -0.3504 -999 -1.4334

221 -1.4187 -1.3125 -1.3966 -1.4082

231 -1.3795 -1.3888 -1.3933 -1.3349

241 -1.2802 -1.3368 -1.331 -1.256

251 -1.2591 -1.2137 -1.2577 -999

311 -1.1025 1.2962 -999 -1.1077

321 -1.1382 -1.0879 -0.9931 -1.1165

331 -1.3596 1.3364 -1.3253 -1.2981

341 -1.3031 -1.2464 -1.3605 -1.2603

351 -1.164 -1.2971 -1.2336 -999

411 -1.017 4.5741 -999 -1.0218

421 -0.9169 -0.5675 -1.008 -1.0124

431 -1.0693 -1.0874 -1.0844 4.6263

441 -1.1677 -1.2238 -1.2256 -1.24

451 -1.2525 -1.2648 4.6406 -999

511 -1.0075 -999 -999 -1.085

521 -0.9692 -999 -0.8749 -1.0242

531 -1.0219 -999 -0.9833 -1.0574

541 0.6902 -999 -1.1348 -1.0999

551 1.3364 -999 -1.2378 -999

The agent moved with PGREEDY that would always use best Q-value for next move. For this experiment, the agent could find better path and it would update the Q-Table more efficiently than random policy because the agent used better path to update the Q-Table. Therefore, the number of steps had been decreased and the balance of bank accounts had been increased. The best path found was fourth run which took 126 steps to finish the task with the bank accounts 290. It did not always get a good path like this one.

1. **In Experiment 2** I used α=0.3, and run the Q-learning algorithm for 6000 steps with policy PRANDOM for the first 200 steps of the experiment, and then switch to PEXPLOIT for the remainder of the experiment.

It has completed 16 runs in 6000 steps:

run#: 1 --- steps: 550 --- bank accounts: -134

run#: 2 --- steps: 328 --- bank accounts: 88

run#: 3 --- steps: 302 --- bank accounts: 114

run#: 4 --- steps: 206 --- bank accounts: 210

run#: 5 --- steps: 460 --- bank accounts: -44

run#: 6 --- steps: 370 --- bank accounts: 46

run#: 7 --- steps: 254 --- bank accounts: 162

run#: 8 --- steps: 310 --- bank accounts: 106

run#: 9 --- steps: 208 --- bank accounts: 208

run#: 10 --- steps: 342 --- bank accounts: 74

run#: 11 --- steps: 414 --- bank accounts: 2

run#: 12 --- steps: 346 --- bank accounts: 70

run#: 13 --- steps: 336 --- bank accounts: 80

run#: 14 --- steps: 194 --- bank accounts: 222

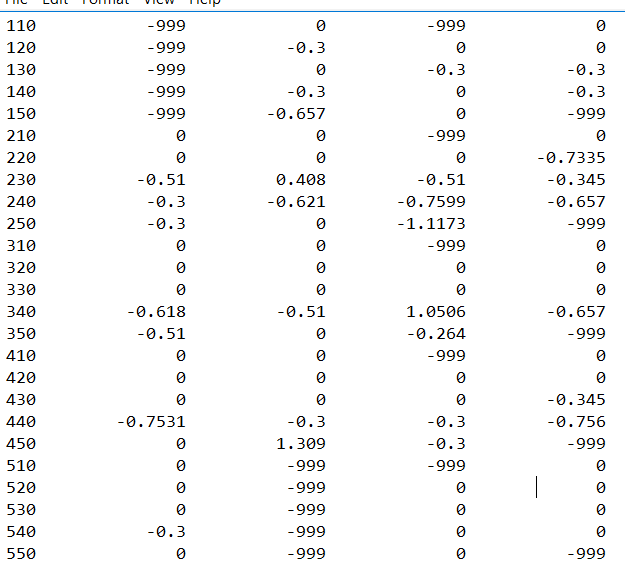
run#: 15 --- steps: 212 --- bank accounts: 204

run#: 16 --- steps: 322 --- bank accounts: 94

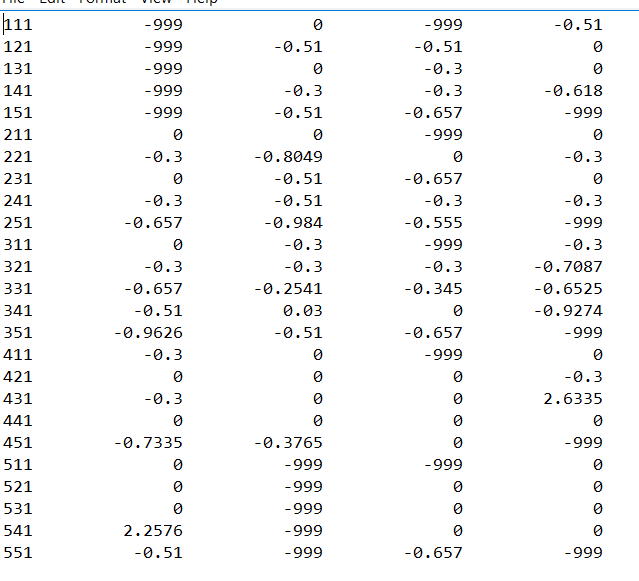
run#: 17 --- steps: 314 --- bank accounts: 102

The Q-Tables I obtained on 182th step (First Drop-off was full):

No carrying block Q-Table:

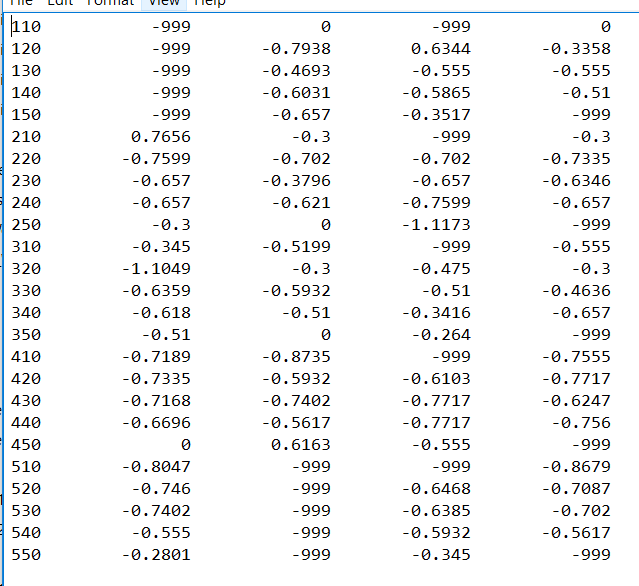


Carrying block Q-Table:

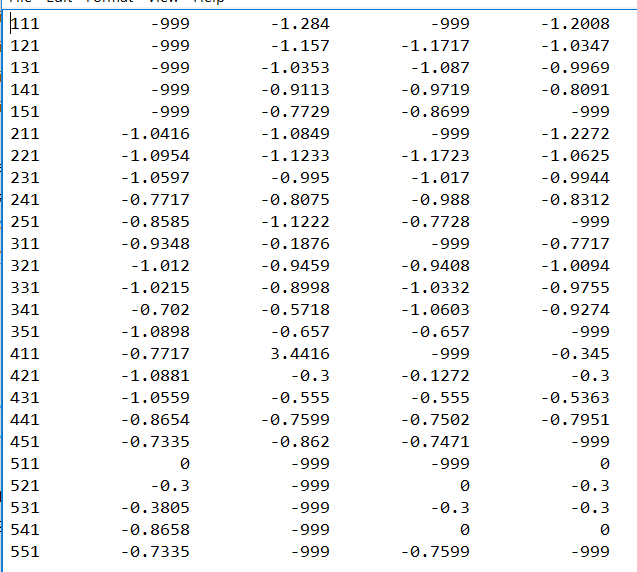


The Q-Tables I obtained on 550th step (First run was completed):

No carrying block Q-Table:

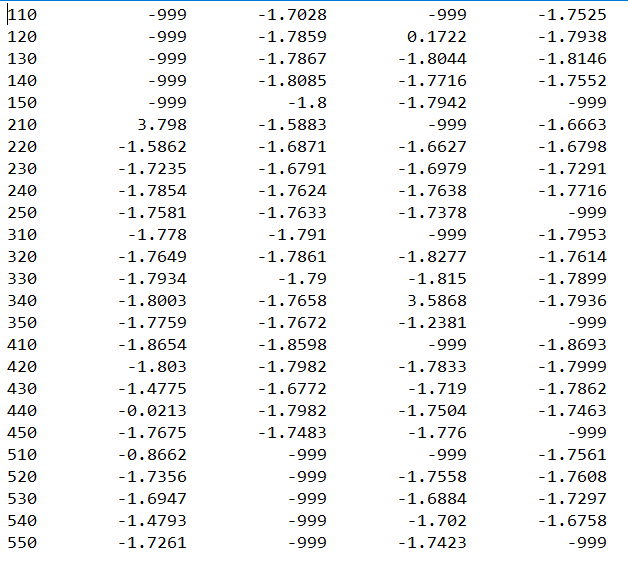


Carrying block Q-Table:

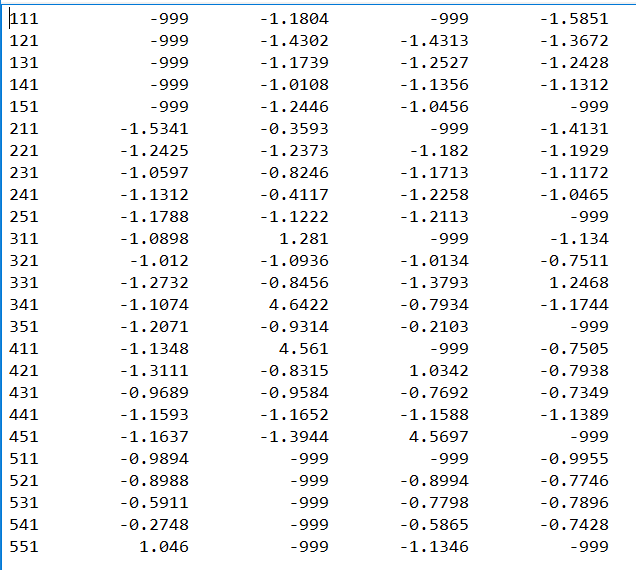


The Q-Tables I obtained on 3000th step (middle of the experiment):

No carrying block Q-Table:

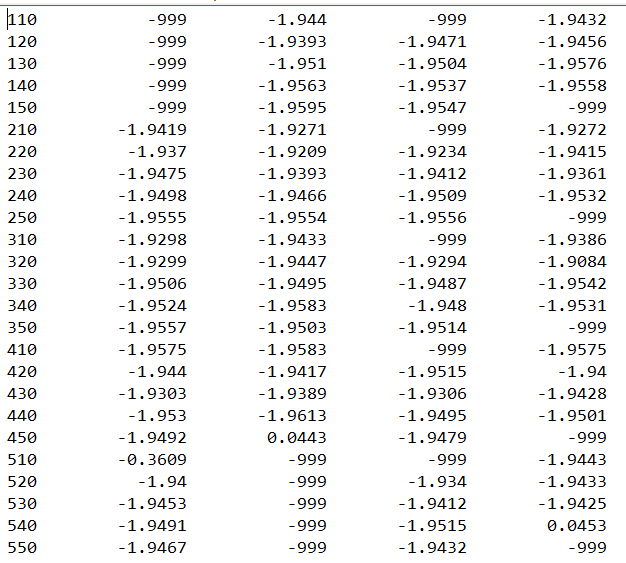


Carrying block Q-Table:

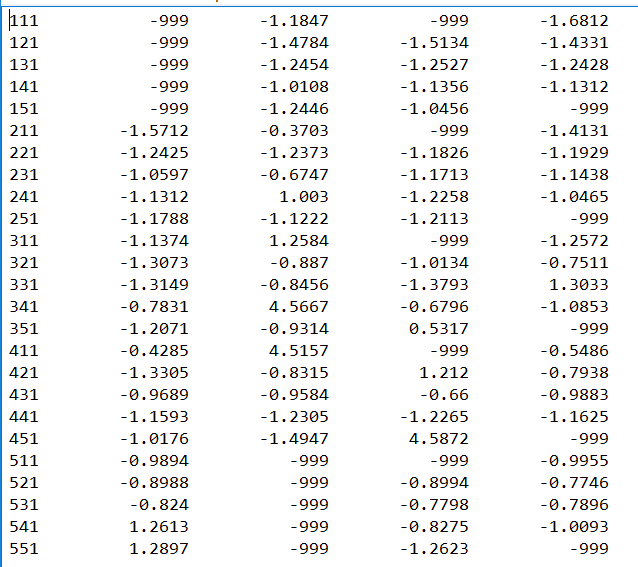


The Q-Tables I obtained on 6000th step:

No carrying block Q-Table:



Carrying block Q-Table:



The performance of this experiment at different states was good, but the agent sometimes could get worse path since we used PEXPLOIT which had 0.15 probability to use a random operator. The Q-Tables were updated well dynamically since that pick-up could be empty and drop-off could be full, they both would be non-applicable in these situations. At that time, it still used same Q-values for next operator. However, it learned fast to find a good path, but it was not perfect. The Q-values on state 451 and 351 would always go left if it was not random because of their two drop-off locations were located on the left. There existed few optimal paths like this. Other states would pick an operator based on the current PD-World.

1. **In Experiment 3** I used α=0.3, and run the SARSA q-learning variation for 6000 steps with policy PEXPLOIT - However, use policy PRANDOM for the first 200 steps of the experiment, and then switch to PEXPLOIT for the remainder of the experiment.

run#: 1 --- steps: 603 --- bank accounts: -200

run#: 2 --- steps: 372 --- bank accounts: 44

run#: 3 --- steps: 370 --- bank accounts: 46

run#: 4 --- steps: 218 --- bank accounts: 198

run#: 5 --- steps: 448 --- bank accounts: -32

run#: 6 --- steps: 332 --- bank accounts: 84

run#: 7 --- steps: 264 --- bank accounts: 152

run#: 8 --- steps: 592 --- bank accounts: -176

run#: 9 --- steps: 610 --- bank accounts: -194

run#: 10 --- steps: 538 --- bank accounts: -122

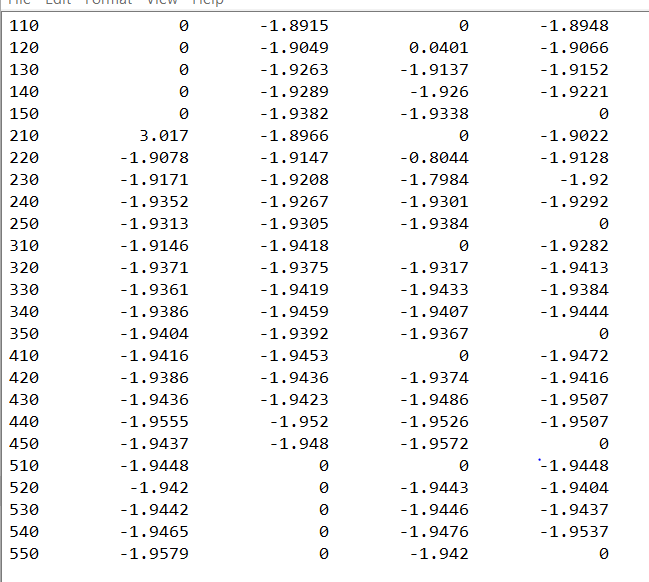
run#: 11 --- steps: 856 --- bank accounts: -440

run#: 12 --- steps: 756 --- bank accounts: -340

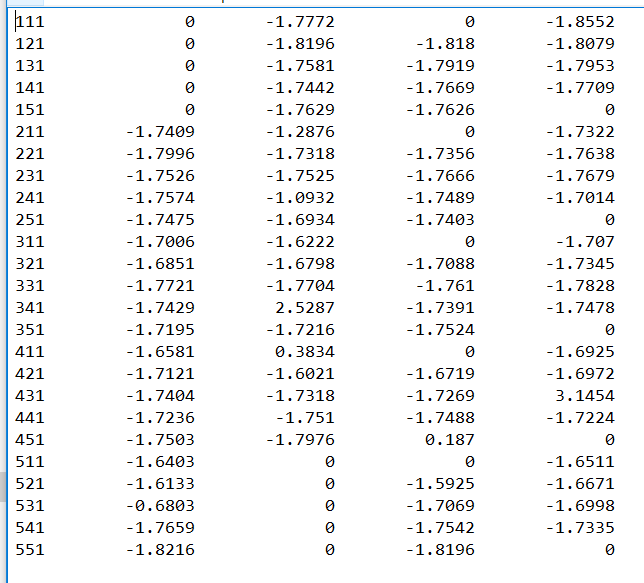
It was possible to get worse result than PRANDOM.

The Q-Tables I obtained on 6000th step:

No carrying block Q-Table:



Carrying block Q-Table:



Compare to Q-Learning, SARSA had more bad results on later runs since it used the value of actual moves. When the agent moved to a low score position more, the agent could have bad result.

Analysis of Attractive Paths (Q-Learning with exploit policy on 3000th step)

No Carrying Block Q-Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -------  ------- -1.7525  -1.7028 | -------  0.1722 -1.7938  -1.7859 | -------  -1.8044 -1.8146  -1.7867 | -------  -1.7716 -1.7552  -1.8085 | -------  -1.7942 ------  -1.8 |
| 3.798  ------ -1.6663  -1.5883 | -1.5862  -1.6627 -1.6798  -1.6871 | -1.7235  -1.6979 -1.7291  -1.6791 | -1.7854  -1.7638 -1.7716  -1.7627 | -1.17581  -1.7378 ------  -1.7633 |
| -1.778  ------ -1.7953  -1.791 | -1.7649  -1.8277 -1.7614  -1.7861 | -1.7934  -1.815 -1.7899  -1.79 | -1.8003  3.5868 -1.7936  -1.7658 | -1.7759  -1.2381 ------  -1.7672 |
| -1.8654  ------ -1.8693  -1.8598 | -1.803  -1.7833 -1.7999  -1.7982 | -1.4775  -1.719 -1.7862  -1.6772 | -0.0213  -1.7504 -1.7463  -1.7982 | -1.7675  -1.776 ------  -1.7483 |
| -0.8662  ------ -1.7561  ------ | -1.7356  -1.7558 -1.7608  ------ | -1.6947  -1.6884 -1.7297  ------ | -1.4793  -1.702 -1.6758  ------ | -1.7261  -1.7423 ------  ------ |

Analysis of Attractive Paths (Q-Learning with exploit policy on 6000th step)

Carrying block Q-Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -------  ------- -1.6812  -1.1847 | -------  -1.5134 -1.4331  -1.4784 | -------  -1.2527 -1.2428  -1.2454 | -------  -1.1356 -1.1312  -1.0108 | -------  -1.0456 ------  -1.2446 |
| -1.5712  ------ -1.4131  -0.3703 | -1.2425  -1.1826 -1.1929  -1.2373 | -1.0597  -1.1713 -1.1438  -0.6747 | 1.1312  -1.2258 -1.0465  1.003 | -1.1788  -1.2113 ------  -1.1222 |
| -1.1374  ------ -1.2572  1.2584 | -1.3073  -1.0134 -0.7511  -0.887 | -1.3149  -1.3793 1.3033  -0.8456 | -0.7831  -0.6796 -1.0853  4.5667 | -1.2071  0.5317 ------  -0.9314 |
| -0.4285  ------ -0.5486  4.5157 | -1.3305  1.212 -0.7938  -0.8315 | -0.9689  -0.66 -0.9883  -0.9584 | -1.1593  -1.2265 -1.1625  -1.2305 | -1.0176  4.5872 ------  -1.4947 |
| -0.9894  ------ -0.9955  ------ | -0.8988  -0.8994 -0.7746  ------ | -0.824  -0.7798 -0.7896  ------ | 1.2613  -0.8275 -1.0093  ------ | 1.8297  -1.2623 ------  ------ |

This table shows there exists some local optimal paths especially for the agent moves with a block. When the agent got closer to drop-off locations, it would know how to find nearest drop-off location based on the prior knowledge.

**Issues and bugs that I encountered during the implementation:**

Bank accounts were always decreased by reward -1, hardly to find pick up and drop off location. The path that agent has passed was worse than random case. Using PRANDOM normally took 600 – 700 steps. Using PEXPLOIT method normally take 2000 -3000 steps on first round, then never terminated on later rounds if I did not set the maximum number of looping times. The agent got stuck on top left with strange loop. Because I did not add the variable “carryingBlock” for next state in some situation which misled the agent to wrong direction then stuck on the strange loops such as top left corners.

The agent was scared and stayed on the start point after one termination using SARSA with PEXPLOIT. Because it got non-applicable operator which was -999 by code error, then the Q-Table was updates with this negative number like there was a monster on (5, 1). After another drop-off location was full, the agent just got back to start point after several hundred moves.

**Conclusion:**

Since the learning rate and discount rate were fixed, so the results were only based on those three policies. By doing these experiments, I have learned more about how the reinforcement learning works and solves the real-world problems.

The best path was explored by the agent took 126 moves with the bank account of 290 using Q-Learning with PGREEDY that always picked max Q-Value operator for next move. Q-Learning and SARSA was similar when the agent used PEXPLIT that has 0.15 probability to pick an operator randomly. After I ran many times of each experiment, the result of the PEXPLOIT was vary since that random probability. However, the PGREEDY for Q-Learning was more stable to get a good enough path, but it was based on first 200 random steps.

The only difference between Q-Learning and SARSA is on-policy and off-policy. SARSA uses the values what the agent actual moves to update the Q-Table with the function and Q-Learning uses the max of next possible operators.

SARSA normally found a good path on second or third run, but it usually got scared like a baby when it had a bad position once on later runs, then it would be so hard to overcome and beat the frustration.

The system not always get a better result after it solved a few PD-world problems because there are four pick-up and two drop-off locations and the agent uses the Q-Table with 50 states only. The Q-Values cannot be update on time, they must be learned after the agent know that drop-off locations are full or pick-up locations are empty. The agent sometimes wastes more moves for learning this. After two different runs of each experiment, the learning curves that I have showed on the above were similar, but the results were different. Therefore, I think that the agent can learn it and solve this kind of problem very well if I can adjust this traditional algorithm.