Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

# Group 7 Report

## 1 Introduction

Reinforcement learning is a subfield of artificial intelligence and machine learning. Algorithms featured in reinforcement learning are popular among problems that have no existing data, or issues that are not uniform, like speech recognition. In this project, we will use reinforcement to learn and adapt “promising paths” in our grid environment.

Throughout this report we will cover basic reinforcement learning concepts such as utilities, policies, learning rates, discount rates and their interactions. We conducted five experiments using different parameters and policies, and displayed our results using visualization techniques. We have designed and programmed an agent that must explore and learn in an initially unknown environment while adapting to certain changes. The leader and MVP of this group project, voted on my all members, is without a doubt Vincent Poon.

## 2 Project Description

The goal of this project is to have an agent move a total of 15 blocks from three pickup locations and deliver them to three drop off locations. The environment is a 5x5 grid with the pick up locations at cells (1,1), (3,3), (5,5), the drop off locations at cells (5,1), (5,3), (2,5), and the agent starting at (1,5).

There are six operators, or steps, the agent can take: North, South, East, West, Pickup and Dropoff. North, South, East, West are applicable in each state, and move the agent to the cell in that direction as long as it is within the grid. Pickup is only applicable if the agent is in an pickup cell that contain at least one block and if the agent does not already carry a block. Drop off is only applicable if the agent is in a drop off cell that contains less than 5 blocks and if the agent carries a block. Each operator is assigned a reward to help determine the q-value. Applying north, south, east, west is -1. Picking up a block from a pickup state is +13. Dropping off a block in a drop off state h operator the agent takes next is decided by one of three policies. The q-values used in the policies will be calculated using either Q-learning or SARSA.

1. PRandom: If pickup and dropoff is applicable, choose this operator; otherwise, choose an operator randomly.
2. PExploit: 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 0.8 and choose a different applicable operator randomly with probability 0.2.
3. PGreedy: 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).

Lastly, the performance of the agent will be measured by how many times it reaches the terminal state, number of operators applied to reach a terminal state from the initial state and the total reward. We will want to maximize our reward while keeping the number of operators applied to a minimum.

## 3 Q-Learning and SARSA algorithms

Q-Learning is an off-policy learning algorithm and geared towards the optimal behavior although this might not be realistic to accomplish in practice, as in most applications policies are needed that allow for some exploration. SARSA, on the other hand, uses the actually taken action for the update and is therefore more realistic as it uses the employed policy; however, it has problems with convergence. We will briefly discuss the update formulas used and will compare these two algorithms later using the experiments.

Q-Learning: Q(a,s) = ( 1 - alpha ) * Q(a,s) + alpha * (R(s) + gamma * max(Q(a',s')))
The last term in the equation selects the action that has the max q-value, this indicates some form of greedy behavior is involved in Q-Learning.

SARSA: Q(a,s) = ( 1 - alpha ) * Q(a,s) + alpha * (R(s) + gamma * Q(a',s'))
The last term of SARSA take the one with the operator that will be applied next instead of map q-value like Q-Learning

## 4 Set-up/ Running the code

The project was written in Python using PyCharm. The interpreter that needs to be installed are pygame, copy and random. The remaining imports are all written as part of the project.

In order to execute the five experiments, select Experiment1.py Experiment2.py … Experiment5.py. When the file is ran, it will open a window with a visualized graph of the world for every 2000 iterations of agent steps. The window must be closed to view the subsequent graph.

## 5 Implementation

To create our environment, we created a class Node, which holds information about whether it is a pick up or drop off location, its block count, and q-values for the north, south, east, west operators. We then mapped the node in a 5x5 matrix.

Since different paths are attractive for agents who hold a block and for agents who do not hold a block, we decided to make two matrices or worlds to depict this. Every time the agent picks up a block it will switch to the world where attractive paths lead to drop off locations. When the agent drops off a block, it will switch to the world where attractive paths lead to pick up locations.

In our visualization windows, we will show the world when the agent is not holding a block on the left, and the world where the agent is holding a block on the right. The agent is shown as a blue/pink dot in the middle of the tile where he is currently located. Blue indicates he is not holding a block, and pink he is. Each tile/square is divided into 4 triangles representing the North, East, South and West operators, each having a q-value. A triangles colored green indicate a positive q-value, while red indicates a negative q-value. The brighter the green, the higher the q-value. The brighter the red, the lower the q-value. Dark colors, or the color black, indicate q-values close to zero. Using this color coding, we can identify attractive paths by looking for green areas.

Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

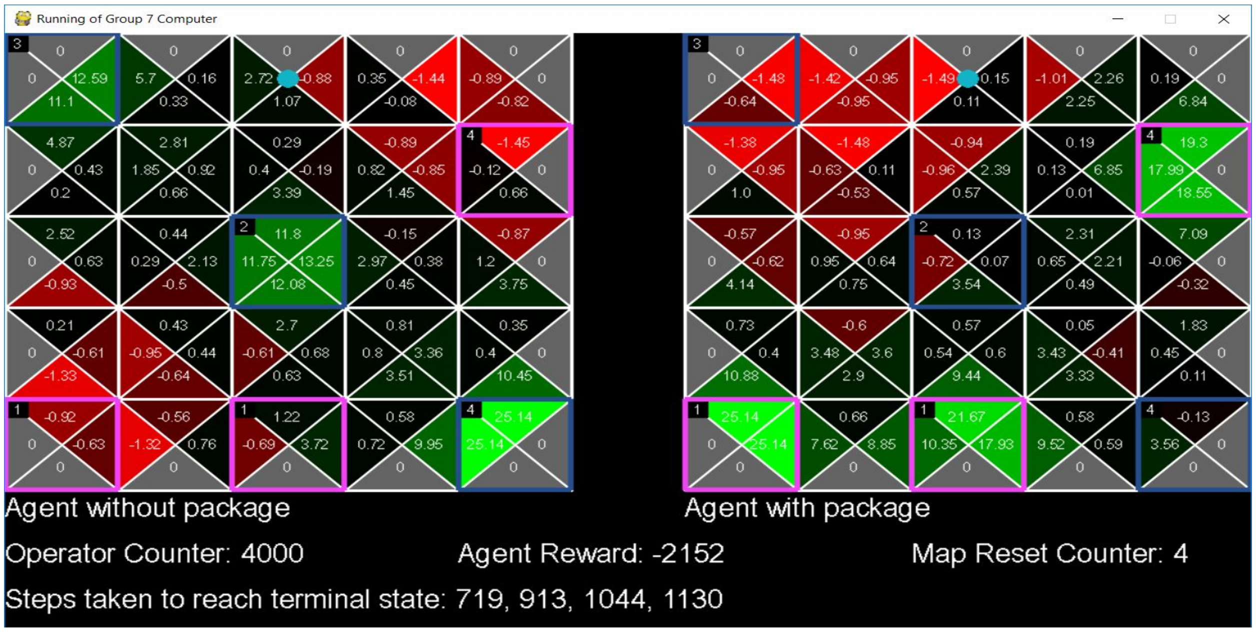
## 6 Experiment Results and Analysis

The agent will begin at position (1, 5), the top-right corner of the world where the agent does not have a package. The default alpha (learning rate) = 0.3, gamma (discount factor) = 0.5. All q-values begin as 0 for both worlds. The world will reset the block count of pick up and drop off locations when the agent completes all of his deliveries. The q-table will NOT be reset during an experiment.

**Experiment 1:**

In this experiment, we will use the Q-learning algorithm. We ran 4000 steps with policy PRANDOM, followed by 4000 steps with PGREEDY with learning rate **α** = 0.3.

E1 Figure 1: After 4000 steps of the first run



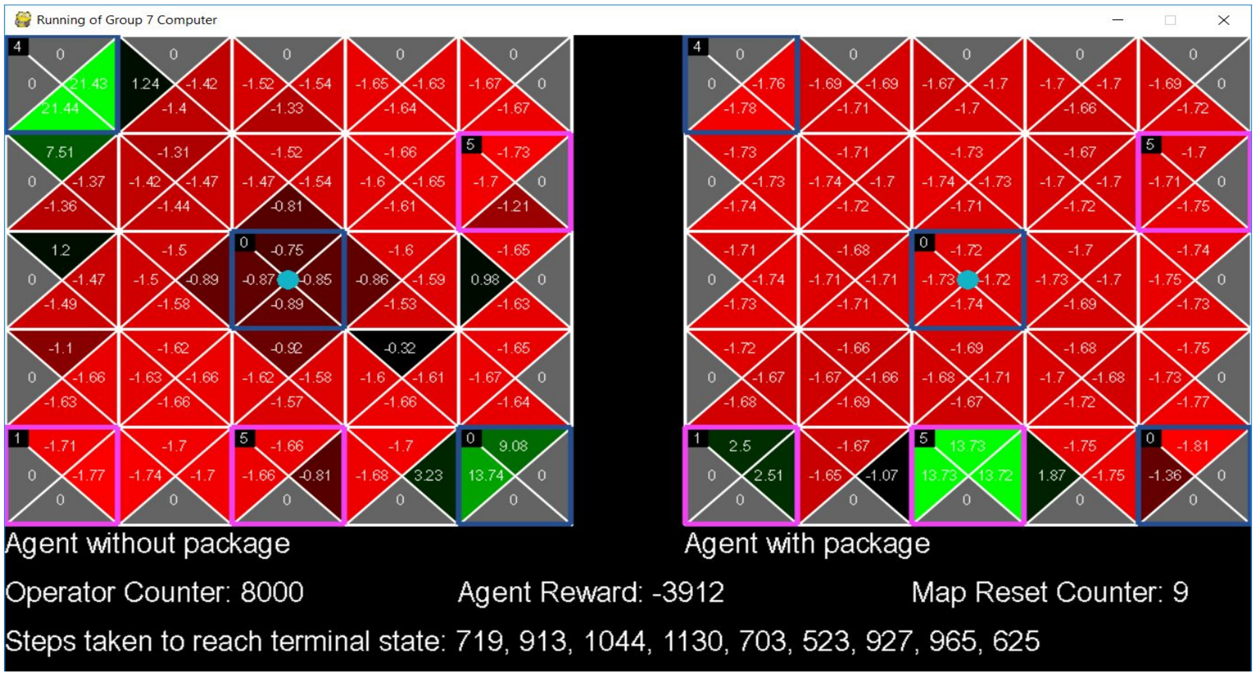
```
Agent without package                                   Agent with package
Operator Counter: 4000          Agent Reward: -2152          Map Reset Counter: 4
Steps taken to reach terminal state: 719, 913, 1044, 1130
```

Experiment 1 Figure 1 (E1F1) shows the first 4000 using policy PRANDOM. On the right grid where the agent is not holding a package, we see that the paths on or near pick up are green. This is expected because the agent receives a relatively huge reward for picking up a package at the pickup locations. Surrounding the green pick up area, we see a dark area, followed by a few red triangles at points farthest away from the pickup location. This is because the further you are from the pick up/drop off state, the less attractive the state will be as they do not provide any direct reward. The right grid shows a similar pattern around drop off locations.

We can see that indeed, applying policy PRANDOM for exploration indeed generated a decent idea of which section of the map is the most attractive, namely the pick up and drop off location on the map. At this point, the agent has reached termination 4 times with the agent reward totaling -2152 points. The number of steps taken to reach each terminal state are 719, 913, 1044, 1130. We noted the steady increase in steps from 719 to 1130 but since the operators were chosen randomly, there is not much for us to say about it.

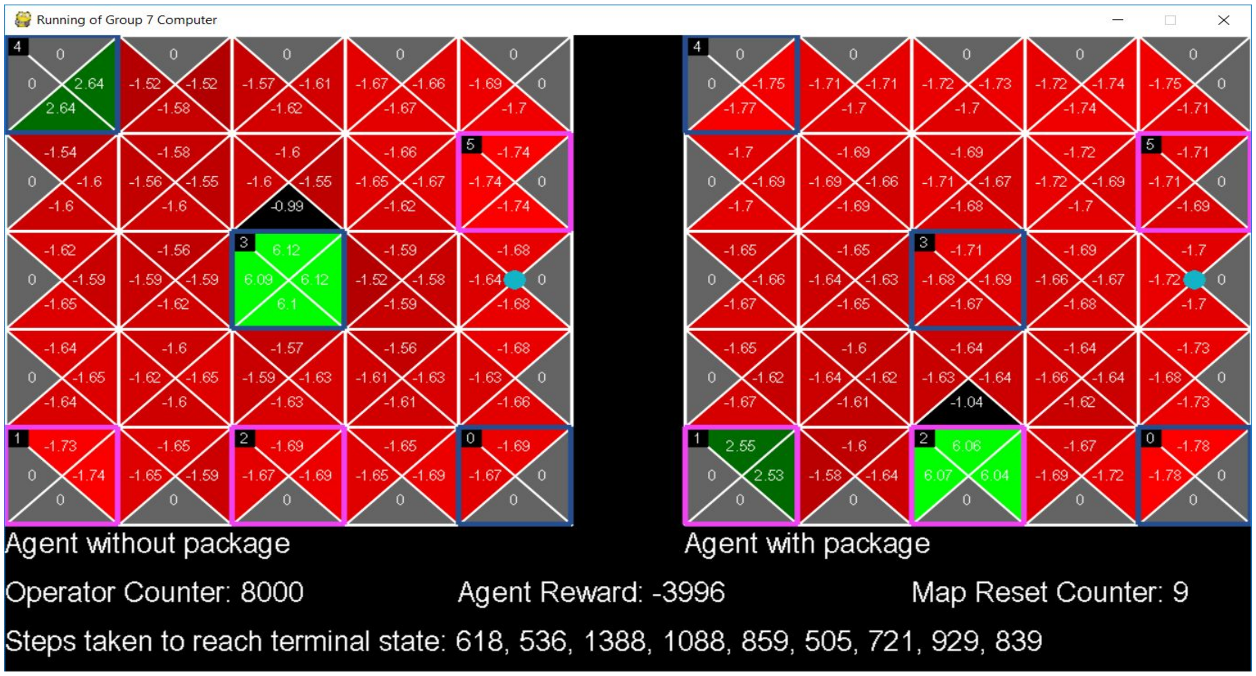
Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

E1F2: Completion of the first run



```
Operator Counter: 8000          Agent Reward: -3912          Map Reset Counter: 9
Steps taken to reach terminal state: 719, 913, 1044, 1130, 703, 523, 927, 965, 625
```

E1F3: Completion of the second run



```
Operator Counter: 8000          Agent Reward: -3996          Map Reset Counter: 9
Steps taken to reach terminal state: 618, 536, 1388, 1088, 859, 505, 721, 929, 839
```

Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

First run:
Agent reward: -3912
Steps taken to reach terminal state: 719, 913, 1044, 1130, 703, 523, 927, 965, 625

Second run:
Agent reward: -3996
Steps taken to reach terminal state: 618, 536, 1388, 1088, 859, 505, 721, 929, 839

E1F2 shows the completion of the first run using PGREEDY after another 4000 steps. We first notice a huge change by the dominating red color on the grids, even in one of the drop off locations when the agent does have a package. The reason for this could be that the agent attempted to visit states that no longer provide reward. For example (3, 3)( empty pick up),(2, 5)(full drop off) even though they are an attractive state with a high q-value from previous updates. The agent will continue to visit those states until their Q-Value drops to lower than other states.

We see that the agent reaches a termination state a total of 9 times with a rewarding total of -3912. An important point to notice here is that the reward deficit only increased by -1760 while the agent has reached the termination step five more times. This means the agent was much more efficient at reaching a terminal state. The number of steps taken to reach each terminal state are 719, 913, 1044, 1130, 703, 523, 927, 965, 625. There is a slight decrease in the range of values from PRANDOM to PGREEDY.

If we compare the two runs we notice several similarities. Grids from the first run, E1F2, are colored the same as grids in the second run, E1F3. We see that the agent increased his iterations of terminal from 4 times to a total of 9 times in E1F3, with the reward deficit increasing from -2040 to -3996. This means that the reward deficit increased by -1956 while reaching a terminal state 1 extra time. We see a form of consistency between these two runs and assume that we would see similar results in further runs.

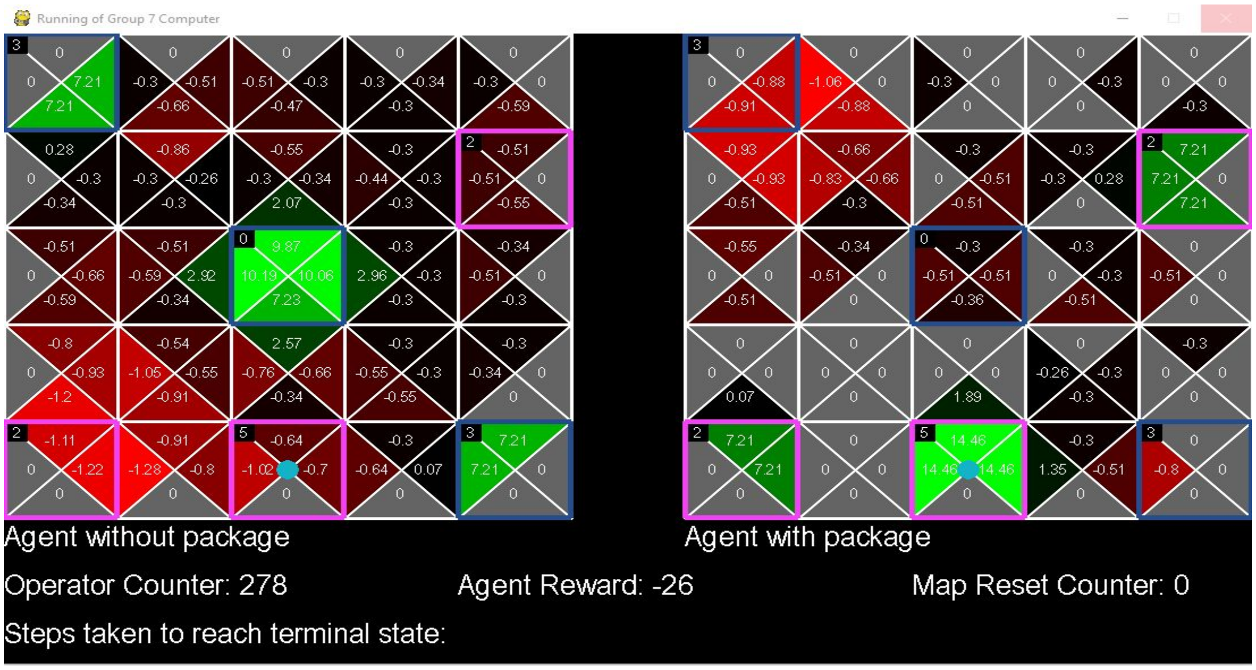
Establishing the q-table values using PRANDOM was ideal for exploration. Once all the possible paths are taken and their q-values are known, there is less need for investigation. Thus, changing the policy to choose the operator with the highest q-value, PGREEDY, showed improvement for the agent reaching a terminal state. The second run displayed the consistency in this improvement.

Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

**Experiment 2:**

In this experiment we will use the Q-learning algorithm. We ran 200 steps with policy PRANDOM, followed by 7800 steps with PEXPLOIT.

E2F1: When the first drop off location is filled for the first run



Agent without package

Agent with package

Operator Counter: 278

Agent Reward: -26

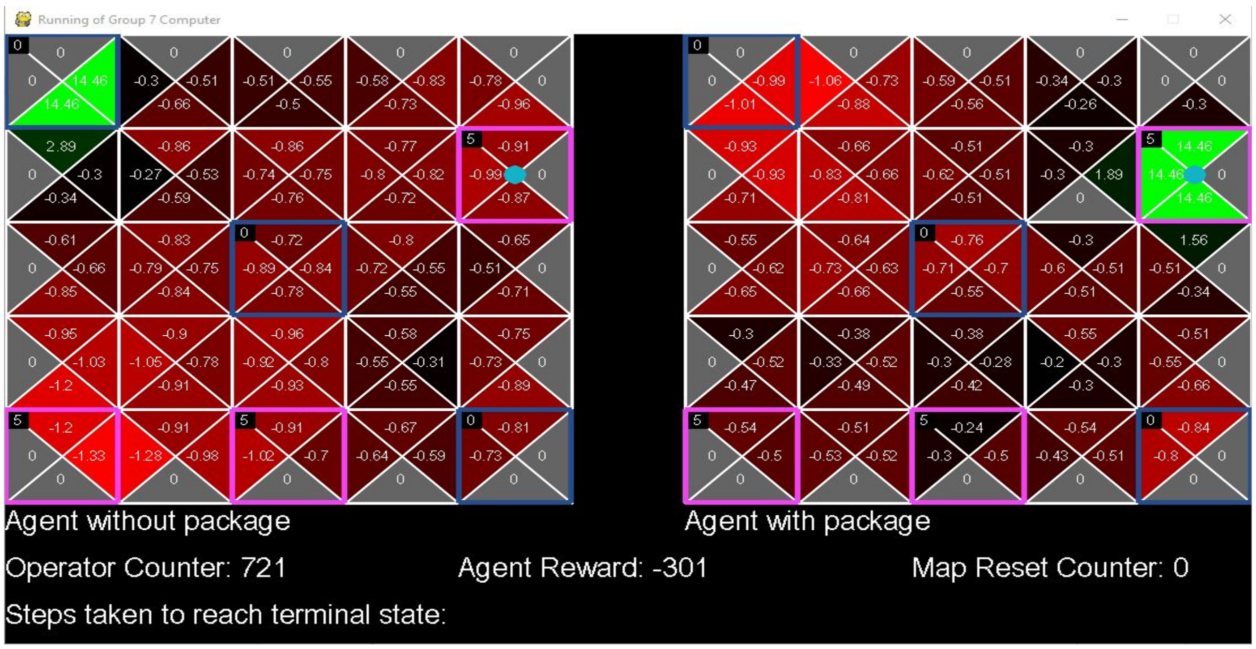
Map Reset Counter: 0

Steps taken to reach terminal state:

In E2F1 the grids for the first drop off location are filled from the first run. The operator count is stopped at 278 and the agent reward is -26. The first 200 steps were run with PRANDOM, and the following 7800 steps after were run using PEXPLOIT. The right grid is not completely filled or explored because the agent spent most of its time without a package during PRANDOM followed by PEXPLOIT traversing the most attractive path rather than exploring. The pick up locations on the left grid and the drop off locations on the right grid are all green as expected, with red areas at points furthest away.

Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

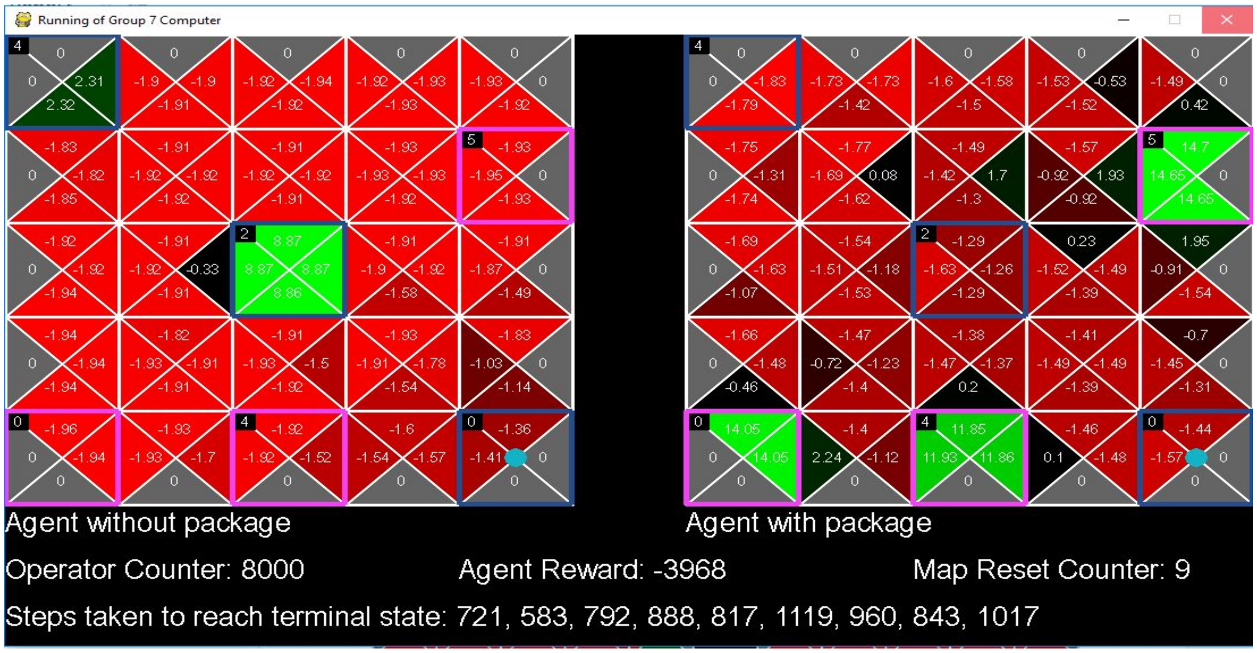
E2F2: The first terminal state is reached in first run



```
Agent without package                                    Agent with package
Operator Counter: 721          Agent Reward: -301          Map Reset Counter: 0
Steps taken to reach terminal state:
```

In E2F2 the first terminal state is reached in the first run. It took 721 steps and the agent reward is -301. It took a total of 521 steps with policy PEXPLOIT, following PRANDOM, to reach a terminal state. The grids have turned mostly red with only one pick up location and drop off location green. We know that these two locations were the last to be visited to deliver the final block because we can see that the agents final location. The other nodes that are shown red are those that were emptied/filled much earlier. Also, in the right grid, almost all the colors have been filled, due to the 20% random operator chosen in PEXPLOIT.

E2F3: Completion of first run



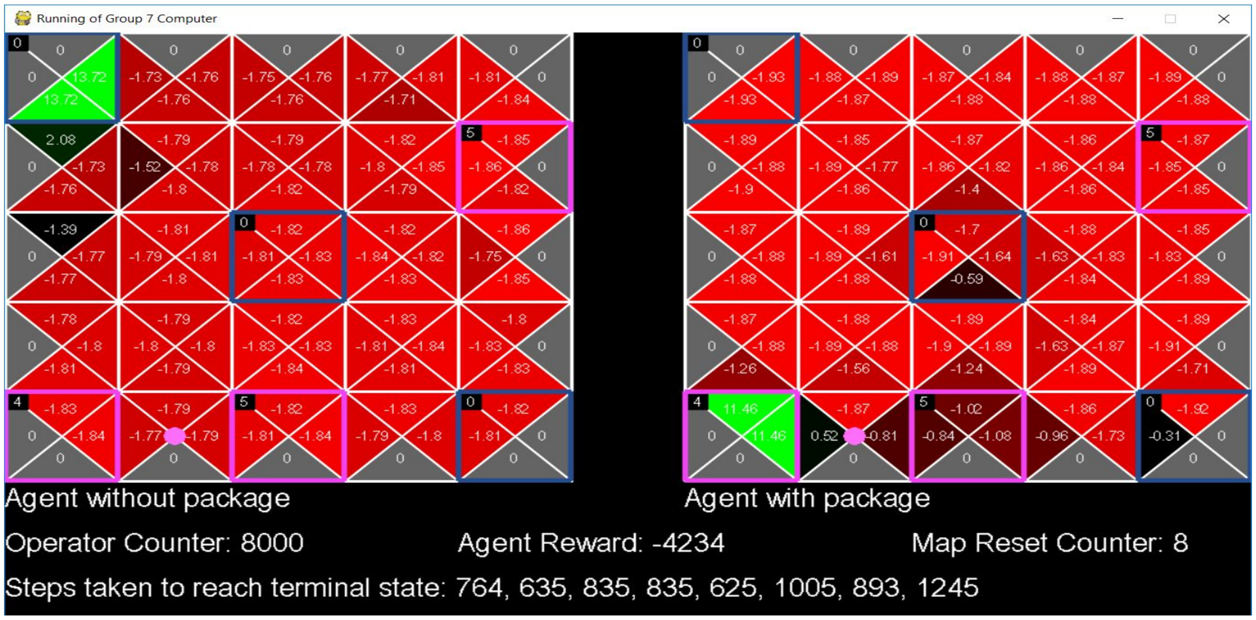
```
Agent without package                                    Agent with package
Operator Counter: 8000         Agent Reward: -3968         Map Reset Counter: 9
Steps taken to reach terminal state: 721, 583, 792, 888, 817, 1119, 960, 843, 1017
```

Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

First run:
Agent reward: -3968
Steps taken to reach terminal state: 721, 583, 792, 888, 817, 1119, 960, 843, 1017

E2F3 is the completion of the first run. There is a dominating red color on the grids. On the right grid where the agent is holding a package, we see dark operators leading to the drop off locations depicting the attractive paths. The agent reached a terminal state a total of 9 times with the reward totalling -3968 points. Looking at the steps taken to reach each terminal state, the results show varying numbers ranging from 583 up to 1119. The numbers go up from 817 to 1119 and back down to 960, not showing as much consistency as we would have thought. Overall, we see that the steps taken increased, slowing down performance. We believe that there are some flaws in using Q-Learning to select q-value for the maximum reward for a long period of time.

E2F4: Completion of second run



Operator Counter: 8000          Agent Reward: -4234          Map Reset Counter: 8
Steps taken to reach terminal state: 764, 635, 835, 835, 625, 1005, 893, 1245

Second run:
Agent reward: -4234
Steps taken to reach terminal state: 764, 635, 835, 625, 1005, 893, 1245

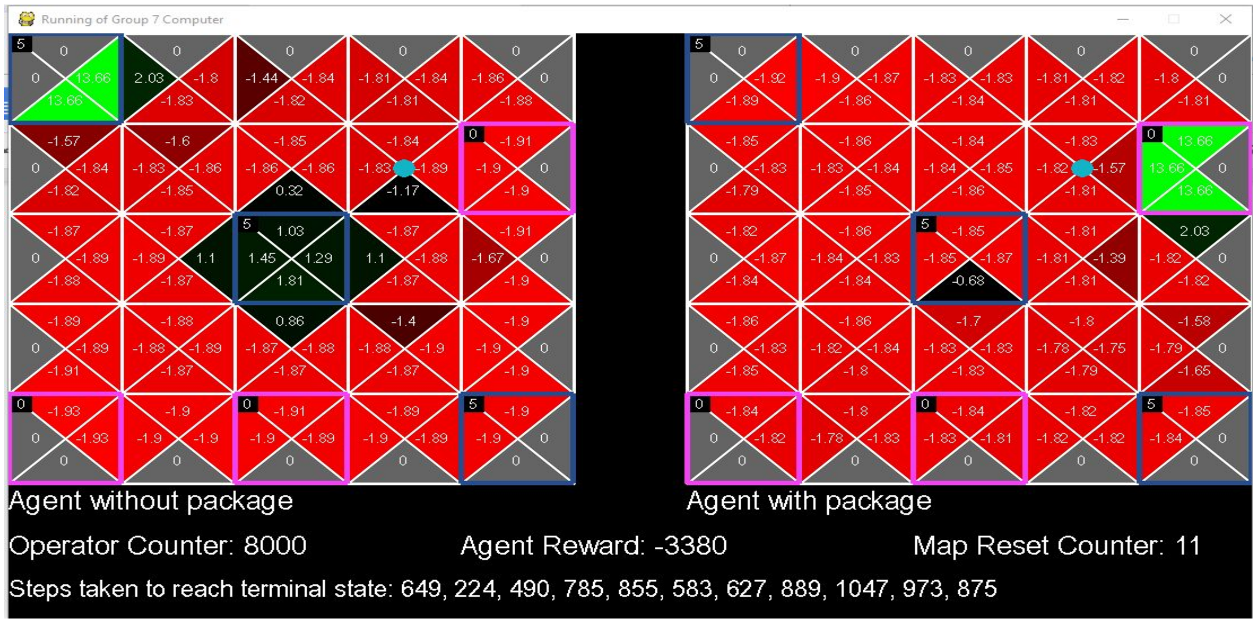
E2F4 shows the completion of the second run. All the pick up locations are empty and the drop off location has one remaining slot open at the only green location at (5,1). The agent holding the package is one step away from completion, depending on if the PEXPLOIT policy chooses the highest q-value or if the correct path is chosen randomly. The second run is similar to the first one in regards to the bright red colors on the grids. The agent reached a terminal state a total of 8 times with the reward totalling -4234 points. In terms of the number of times the agent reaches the terminal state, we can say they are close since the second run was one step away from its 9th. There was only 266 difference in reward as well. Looking at the steps taken to reach each terminal state, we see a similar range of ups and downs. We also see the similar overall increase in steps needed to reach the terminal state.

This experiment showed us how the policy PEXPLOIT worked in a changing state space, where pick up and drop off location might or might not provide rewards. Both runs showed an initial drop in the steps needed to reach a terminal state but gradually increased throughout the experiment.

**Experiment 3:**

In experiment 3, we used two different policies to perform SARSA updates with alpha=.3 and gamma =.5. We began by running 200 steps with the PRANDOM policy and continued to run 7800 steps using PEXPLOIT.

E3F1: 8000 steps



Agent without package

Agent with package

Operator Counter: 8000

Agent Reward: -3380

Map Reset Counter: 11

Steps taken to reach terminal state: 649, 224, 490, 785, 855, 583, 627, 889, 1047, 973, 875

First run:
Agent reward: -3380
Steps for each termination: 649,224,490,785,855,583,627,889,1047,973,875

Second run:
Agent reward -3940,
Steps for each termination: 706,771,711,827,983,845,969,891,1113

The first run produced 11 map resets over 8000 steps. That's an average of 1 completed run every 730 steps. The second run performed slightly worse with 9 terminations.

In comparison to Q-Learning, which the update algorithm is used in Experiment 2, it completed 2 extra times, and the steps needed to complete delivery is on average about 160 steps fewer, indicating us that this is significantly better. Upon further inspection, the steps needed for completion also did not increase as much as the experiment goes on. This is likely due to the nature of SARSA being on policy rather than off policy like Q-Learning, where it successfully avoided some pitfall of being not exploring like Q-Learning this behavior causes it to not learn about the environment and not select some states, that are actually “good” state the agent wants to be in.

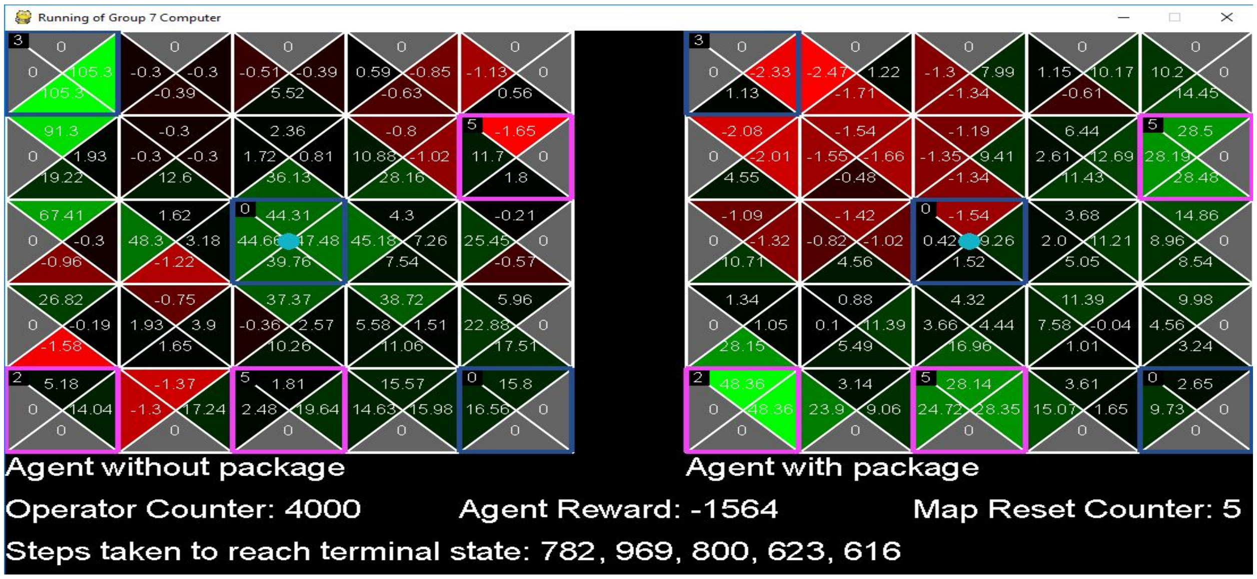
The Q-Table also looks similar for SARSA and Q-Learning, because both ended up in a state that only few rewards were left on the map. For Q-Learning, the highest Q-value is chosen. SARSA chooses the q-value to correspond to the applied operator to update the q-table.

The steps needed to complete delivery for the first few iterations are similar at around 600. As time goes on, SARSA’s average is 800, where Q-Learning becomes 900. SARSA is performing better in a constantly changing world, where the pick-up and drop-off will provide rewards or no rewards depending on the state space

**Experiment 4:**

In experiment 4, we took the exact same steps as in experiment 3, but used a gamma value of 1 instead of 0.5.

E4F1

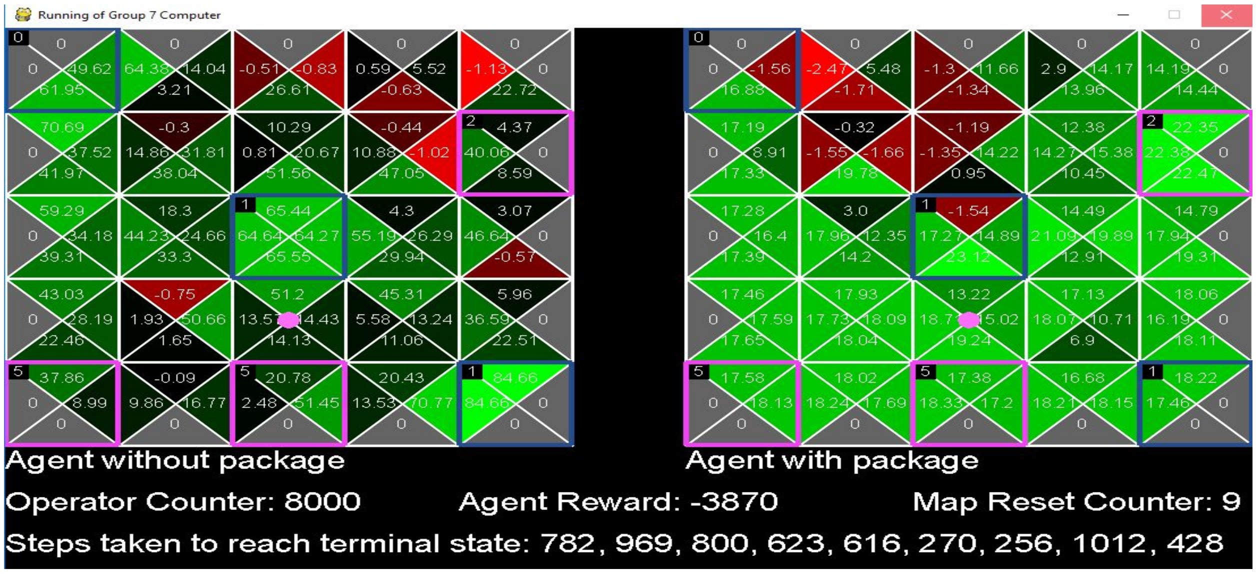


Operator Counter: 4000 Agent Reward: -1564 Map Reset Counter: 5

Steps taken to reach terminal state: 782, 969, 800, 623, 616

Group 7: Logan Fortune, Andy Winata, Travis Lawrence, Joe Bellaish, Vincent Poon

E4F2



Agent without package Agent with package

Operator Counter: 8000 Agent Reward: -3870 Map Reset Counter: 9

Steps taken to reach terminal state: 782, 969, 800, 623, 616, 270, 256, 1012, 428

First run:
Agent reward: -3870
Steps taken to reach terminal state: 782,969,800,623,616,270,256,1012,428

Second run:
Agent reward: -4304
Steps taken to reach terminal state: 1189, 738, 1454, 1015, 1118, 518, 483, 705

On the first run, the map was reset 9 times, and on the second it was reset 6 times. Having a larger discount rate seemed to stunt the learning progress somewhat.

In this experiment, we will focus on how the discount factor(gamma) in the equation effects the performances of the agent. We can see that at step 8000, a majority of the map has turned green, where the agent believes all of the state will yield rewards. In contrast to experiment 3, where the majority of the map is red. The agent's action is now choosing the direction of highest reward, instead of the path with the least penalty.

The map turning from red to green is caused by the discount factor in the SARSA equation. The most direct effect the gamma variable will have is on the final variable, Q(a',s'), the utilities of the future state. In turn, gamma affects how the agent sees the future.

The q values are more prevalent in experiment 3 and therefore there are more completed runs on average. Through further observation of experiment 3 and 4, we see that there are better results when the gamma value is lower. In experiment 4 the gamma value is set to 1 and that almost completely negates any negative effect of moving to a non-reward giving state.

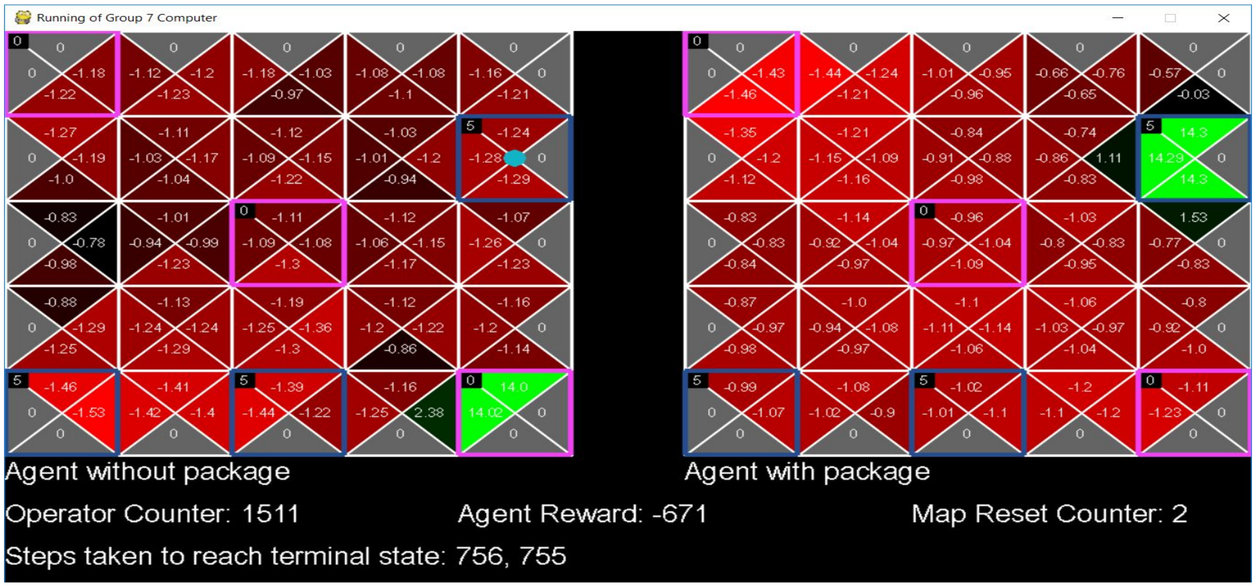
There is a sweet spot of gamma's value before the performance drops. In this experiment, the performance is better when the gamma value is 0.5, instead of 1.0, indicating the ideal gamma is closer to 0.5.

When gamma is 1, the agent thinks all the states is good because it knows it will receive a reward in the future. The penalty aspect is more prevalent when gamma is .5.

**Experiment 5:**

In this experiment we will use the Q-learning algorithm. We ran 200 steps with policy PRANDOM, followed by 7800 steps with PEXPLOIT. This is the same as Experiment 2, except that after the agent reaches a terminal state the second time, we will swap pickup and drop-off locations.

E5F1


Running of Group 7 Computer

Agent without package

Agent with package

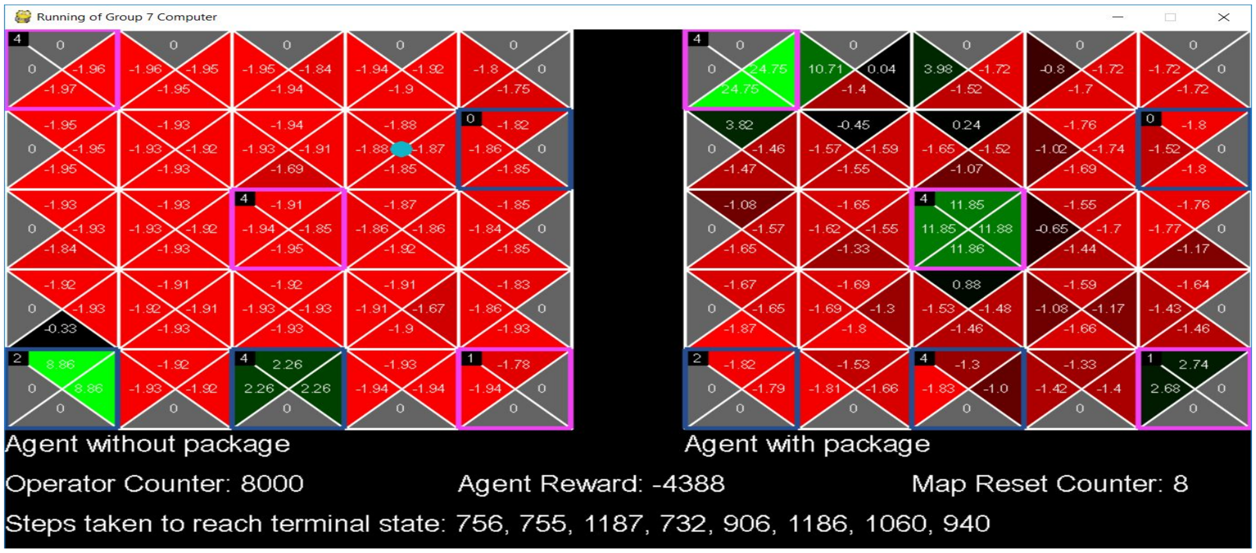
Operator Counter: 1511

Agent Reward: -671

Map Reset Counter: 2

Steps taken to reach terminal state: 756, 755

E5F2


Running of Group 7 Computer

Agent without package

Agent with package

Operator Counter: 8000

Agent Reward: -4388

Map Reset Counter: 8

Steps taken to reach terminal state: 756, 755, 1187, 732, 906, 1186, 1060, 940

First run:
Agent reward: -4388
Steps taken to reach terminal state: 756, 755, 1187, 732, 906, 1186, 1060, 940

Second run:
Agent reward: -4780
Steps taken to reach terminal state: 720, 579, 1339, 960, 1162, 1326, 1352

At step 400, agent reward is -1984, reset counter 4

Experiment 5 performed in the same way as experiment 2, with an alpha value of 0.3, a gamma value of 0.5, and running the PRANDOM policy with Q-Learning for 200 operations followed by 7200 operations using PEXPLOIT. The key difference is, after reaching a terminal state for the second time, the locations of the drop-off spots and pick-up spots were swapped.

While the Q-table is initially washed in red, similarly to experiment 2, the algorithm seems to adapt well after the drop-off and pick-up spots are reversed, particularly in the ex-drop-off point in the center.

In image E5F1, which depicts the algorithm immediately after reaching the termination state a second time, the new pick-up state on the pick-up map is red and drop-off is green, and vice versa on the drop-off map. However, as the experiment continues, the reassigned states revert to their proper colors. In the following figures, some states remains red, but that is due to running out of blocks/space to drop off blocks, meaning no more pick-ups/drop-offs can be performed and no reward can be obtained.

The two runs of the experiment resulted in 8 and 7 resets, respectively, so while it may not be as good as some of the other runs, it is still fairly impressive given that it managed to recover in less than 1300 steps after everything that it learned it knew was changed. However, the runs after the third termination seem to neither increase or decrease in steps consistently, so it may take a while to reach the efficiency first established in PRANDOM.

The results of this experiment show that this reinforcement learning algorithm is capable of effectively adapting to a constantly changing world.

**Overall Analysis of Experiments:**

We found that applying SARSA is better than Q-Learning, and having a better gamma rate will also improve the performance of the agent. Out of all 5 experiments, experiment 3 was able to complete 11 runs, highest out of all 5, which also matches our findings.

Our agent did not get significantly faster after reaching terminal state for the most part. Exception being in experiment 5, that after the pick up/ drop off being flipped, it took a while to adapt to the new state space, then the remaining runs are relatively faster compare to the steps need for the third termination which took some extra steps to readjust

For all 5 experiments both runs generally agrees with the other runs, as the produce similar results. Except for experiment 3, where one of the runs terminated 9, one is 11. Which is a significant amount of differents. However, we are simply getting lucky here on the second run that terminated 11 times, with of the termination took only 224 steps, that is 400-500 steps lower than average and it allows it to complete a few more times.

## 7 Conclusions

When applying reinforcement learning, we should carefully design the exploration and exploitation parameters(alpha and gamma). As exploration provides no direct reward, effectiveness decrease as time goes on. Suggested alternative policy other than a PEXPLOIT with 50% change on exploitation and exploration, we can consider dividing the run into sections and each section must be follow by a different policy. PGREEDY -> PRANDOM - > PGREEDY and so on, to maintain the effectiveness of the agent.

We discovered that SARSA performs better in a dynamic world where reward's location are constantly changing. The higher the gamma, the further the agents sees into the future, however there is a sweets spot that the efficiency will be the best. If the gamma is either below or above (the sweet point) the agent will not behave optimally. Furthermore, even if we change the world when the agent is midway through its learning, the agent can quickly adapt to the new world instead of stopping completely. infact, the performance did not suffer after it adapt to the new world in comparison to the performance before the changes were made.

## 8 Challenges

Prior to starting the project, no one on the team had experience with Python, Pycharm, IDLE environments, or the Pygame library. The group members all had fairly different schedules, so it was difficult to find times that we were all available to meet. A few member of the group had a lot of projects and outside distractions, so they weren't able to contribute much to the project earlier on.

## 9 Contributions

**Vincent**: design and written of the base function for the project, write a template for q-learning and sarsa. Visualization. Patched most bugs. Wrote experiment report.
**Andy**: worked on visualizing the PD world, wrote experiment 5, helped implementing visualization for experiments, wrote report and help debug some of the errors.
**Joe**: worked on visualizing the PD world, wrote and conducted experiment 3, helped on functions required to create the gradient of the colors in the output screen. Various other functions.
**Logan**: Coded Q-Learning and SARSA update functions. Wrote and designed the slides and report. Experiment 1.
**Travis**: Worked on developing gradients for the visualization of the PD world and conducted experiments 2 and 4.
**All additional screen shots are in the screenshot file.**