# Q Learning

Group J
Devin Crane Gustavo Aguilar Nicholas Kaminski

**Introduction**

The implementation of a reinforcement learning algorithm involves many different concepts. At the most basic level these algorithms involve an **agent** with a set of actions on a set of states. The agent must follow a **policy** which takes the current state of the problem and gives the agent some action to take. Each state-action pair has an associated **reward**, which serves as feedback to either encourage or discourage the agent from taking similar actions. While the agent traverses through different states it begins to discover the **utility** of each state, typically expressed as a number. When possible the agent will typically try to move to state with high utilities as the algorithms have been designed such that this process should lead towards the goal. As the agent moves it uses an **update rule** to modify its estimated utility values for each state. Utility is calculated using some evaluation function. The function we use is defined as

$$Q(a,s) \leftarrow (1\text{-}a)*Q(a,s) + a*[R(s',a,s)+ \gamma*\max_{a'}Q(a',s')]$$

where

- Q is the current utility, initially zero
- a is the learning rate
- s is the current state being evaluated
- R is the immediate reward function for reaching state s' by taking action a in state s
- $\gamma$ is the discount rate
- $\max_{a'}Q$ is the max utility obtainable from all valid actions after taking action a in state s

These utilities are used along with some state selection policy in order to navigate the agent's world. Q-Learning is a model-free reinforcement learning technique. An agent learns an action-value function to get an expected utility of taking an action in a given state and following some policy afterwards.

For this project, we implemented a Q-learning algorithm for the PD-World, and also conducted 6 experiments using different parameters and policies. We were instructed to run the experiments assuming the discount rate to always be $\gamma = 0.3$ and that the q values are initialized with 0 at the beginning of the experiment. The experiments were run using various combinations of 3 different policies as follows:

- **PRandom**: If pickup and dropoff is applicable, choose this operator; otherwise, choose an operator randomly.
- **PExploit1**: 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 q-value) with probability 0.65 and choose an applicable operator randomly with probability 0.35
- **PExploit2**: 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 q-value) with probability 0.9 and choose an applicable operator randomly with probability 0.1.

We used different random generator seeds in different runs of the same experiment to obtain different results. For the supplied output, a seed of 115 was applied. State space 2 was used, where reinforcement learning states have the form (i, j, x) where (i, j) is the position of the agent and x is 1 if the agent carries a block and 0 otherwise.

**Experiment Setup**

For each experiment a new alpha (learning rate) and random value are supplied. The alpha value affects how quickly new information is applied to the utility of a state and action. The random value determines which policy is used during execution. A random value of one or greater means the agent with move with complete randomness, but pickup or dropoff a block when applicable. As the random value approaches zero the agent tends to use more exploitation of good utility values rather than exploration of a random path. A gamma value (discount rate) is always set to 0.3 to make evaluating the results a little more simple. This allows us to focus on how different learning rates and policies affect the agent.

Each experiment is run twice. The first run is done with an empty Q-Table and the second run uses the knowledge from the first run. We hope the second run in general will yield a quicker execution of the problem because the agent starts with some knowledge of the utility values. Illegal moves such as moving out of bounds or attempting to drop a block in a non-dropoff location are not allowed. However such moves are represented in the Q-Table as having zero utility.

## Results

| Experiment | Steps [Run 1] | Steps [Run 2] | Steps [Run 3] |
|---|---|---|---|
| 1 | 628 | 831 | |
| 2 | 407 | 531 | |
| 3 | 479 | 328 | |
| 4 | 499 | 400 | |
| 5 | 499 | 400 | 655 |
| 6 | 445 | 363 | |

Table 1. Number of steps for each experiment and each run

## Sample Screenshot

**Experiment 1**

This experiment uses the PRandom policy through the entire process. Additionally, both *alpha* and *gamma* are initialized to 0.3.

The most important thing we noticed in the experiment was the number of steps the agent took to get to the goal state. In the first round of the experiment, the terminal state was reached in the step 628, while in the second round the terminal state was found in step 831. However the q-table of the first run was used to do the second run of the experiment. Although the agent had some knowledge about the world it still ended up taking more time to get all the blocks from the pickup locations to the dropoff locations. The reason behind this is that the agent was choosing the next move randomly, therefore the knowledge represented in the q-table was not useful at all.

Moreover, this experiment is the largest one in terms of iteration steps. Same as before, this is because the agent is moving randomly without using the knowledge in q-table. Therefore, the agent will be moving ignoring learned utility and consequently the agent would take a lot more steps to reach its goal.

**Experiment 2**

For experiment 2 we begin with a PRandom policy because no information is in the QTable. We want to move totally randomly. After 100 steps we change to Policy 1 with the probability of choosing a random option of 35%. On printing out the QTables at iteration 100 we see the fully random nature has made some valid pickup and dropoff operations. The full state at this point shows pickup locations (1, 1) has 3 blocks, (3, 3) has 2 blocks, and (5, 5) has 4 blocks. With this information we can postulate that upon switching to a more greedy policy at this point pickup location (3, 3) will be the first to empty because the agent has earned more reward from that location. We expect this information to be useful when moving to a more greedy policy. The next QTable to be printed out occurs after 27 more iterations. This printout occurs because some location has either been a pickup location and become empty or the location has been a drop off location and has become full. Indeed the full state tell us the following information: pickup locations (1, 1) has 3 blocks, (3, 3) has 0 blocks, and (5, 5) has 4 blocks. Our assumption from before holds for this particular random seed, and we can see from the output the same logic applies to drop off locations. The first run terminates successfully after 407 iterations.

The second run of the agent using learned values from the first run gives some insight in the nature of using this particular version of the reinforcement learning space. Initially again we have the first 100 steps as completely random and some dropoffs have occurred. This particular random seed gave similar results to the first run. However this time the run terminates

successfully after 531 iterations. This would normally be discouraging. However recall that we use random seeds. Therefore the simulation is deterministic and not actually random. Currently the seed used is 115 but after changing the seed to 1515 we find entirely different output. With the new seed the first run terminates after 521 iterations while the second run terminates after 379 iterations.

**Experiment 3**

After 100 steps of PRandom, this time it switched to PExploit 2, with an alpha value of 0.3. The first run took 479 steps, and the second run only took 328 so keeping the q-table, rather than starting it over from scratch again definitely made a difference.

We noticed that in the second run, the pickup and dropoff locations are higher than the first run, 12.6 vs. 10.8, and that the paths to and from are also more pronounced with slightly more negative values. This is probably due to having the q-table between runs.

In order to analyze what was learned at different stages of the experiment as requested, we changed it to print out the q-table every 100th iteration. After the first hundred, it had found 2 pickup locations and 2 drop off locations, which makes sense as it was in pure exploration mode and so wouldn't be picking just one path for the first 5 blocks. Each pickup and dropoff location also had the same utility, with the exception of one pickup, that had 2 blocks retrieved instead of just the 1 the others had.

By the 200th iteration, which is now only 30% random, all pickup and dropoff locations had been discovered, one pickup completely drained (3, 0, 4) and one drop off almost completely full (1, 2, 4). This was expected, as the algorithm is targeting the more helpful path 70% of the time and so is a bit faster.

By the 300th iteration, it had slowed down a bit, having only picked up and dropped off 2 additional blocks, bringing the pickup count to (2, 0, 3) and the drop off count to (2, 2, 5). However, this seems reasonable, as it would be in the middle of adjusting to a pick up location being emptied and a drop off location filling up, working to establish new paths.

The next 100 iterations show the algorithm picking up speed, going from having 5 blocks left to pick up to just 2 (0, 0, 2), and just one more drop off location to fill (3, 5, 5). The last 79 steps finishes off the run.

The second run finishes the process about 31% faster, which was expected given that the second run was able to maintain the q-table.

**Experiment 4**

After 100 steps of PRandom, it again switched to Policy 2, with an alpha value of 0.5 this time. The first run took 499 steps, and the second run only took 400. The higher learning rate slowed it down by 4% on the first run and 18% on the second run when compared with Experiment 3, which was not expected. We suspect this is because of the completely Random policy we began the experiment with.

The results at each 100th iteration for the first run were the same as for Experiment 3. The second run is shown below in the following format (p1, p2, p3) (d1, d2, d3):

| | Experiment 3 | Experiment 4 |
|---|---|---|
| Iteration 100 | (3, 3, 5) (0, 1, 2) | (5, 4, 3) (1, 1, 0) |
| Iteration 200 | (3, 0, 3) (2, 4, 3) | (4, 1, 0) (3, 5, 2) |
| Iteration 300 | (0, 0, 0) (5, 5, 4) | (1, 0, 0) (5, 5, 4) |
| To the end (E4 took 74 extra steps) | (0, 0, 0) (5, 5, 5) | (0, 0, 0) (5, 5, 5) |

**Experiment 5**

Experiment 5 does three runs of the simulation with the third run switching the pickup and dropoff locations. The alpha value is 0.5 and like all other experiments it begins with PRandom policy for the first 100 steps followed by Policy 2. The first run of the experiment is just like experiment 4 so we expect the same outcome. This is indeed the case as both complete the first run after 499 steps, and both complete the second run after 400 steps. Now experiment 5 switches the pickup and drop off locations while keeping the information from the previous runs. This particular implementation of the reinforcement learning space using agent row, agent column, and block tends to be weak in this particular experiment. We can see this clearly is the case as the third run terminates successfully after 655 iterations. This tells us the agent is given

completely incorrect information and must do a sufficient amount of iterations to learn the new correct paths.

**Experiment 6**

In this experiment we used the PExploit 1 policy. The *alpha* and *gamma* values are set to 0.5 and 0.3, respectively. This experiment was expected to reach the goal state faster than the experiments with a lower learning value (i.e. experiment 2).

In fact, the number of steps this experiment took in the first run was 445, and the second run scored 363. Naturally, the number of steps is reduced from first run to second run due to the fact that the agent is using some knowledge from the previous run, that is, the q-table of first run.

Once we run the first round of iterations, we noticed that the q-table had some of the pickup and dropoff locations with values near to 13, but not exactly that number. While other action of the same location were very close to zero. This is because the agent could have depleted the blocks earlier than other pickup locations, and then the number tended to converge at the maximum utility value that location can give —which is 13 points. Other pickup and/or drop off locations are a little bit further to converge to the number mentioned before, but an infinite number of iterations would make this occur.

Notice in the q-table after new pickup locations are assigned. The utility values of the previous pick up locations are not destroyed as a result of the change. The algorithm realizes these old locations are no longer valid pick up locations therefore the utility values of the locations themselves are not touched. However the neighboring locations still may appear to be attractive due to the large utility value. These neighboring locations will decrease further in utility as time passes as visiting them continues to yield negative reward. This allows the agent to eventually learn new pick up locations.

**Conclusions**

Experiment 1 is a baseline to judge other experiments. This is because the experiment is always random and therefore ignores utility. The amount of steps taken for all other experiments is substantially lower than the amount of steps taken by experiment 1. This fact supports the claim that the algorithm is implemented correctly and that the policies are working. Experiment 2 and experiment 3 are very similar except for the policy used. Experiment 2 uses PExploit 1 and therefore behaves slightly more randomly. Again with this particular random number seed we see the more random policy perform better with no information. Experiment 2 does better in the first run but Experiment 3 does better in the second run. This is because Experiment 3 is able

to use the learned values more in the second run. Experiment 4 is the same as Experiment 3 but with a higher alpha value. According to the results for this particular random number seed, 115, Experiment 3 performed better with the lower alpha. Recall that a higher alpha (learning rate) causes the agent to prefer recent knowledge. High learning rates also have a tendency to get stuck if goals change and cause longer simulations to occur. Experiment 5 performed exactly as Experiment 4 because all parameters are the same. The third run is different with the pickup and drop off locations are switched. In this case the agent has complete opposite knowledge of true values. For example when the agent has no block the utility values will pull it towards a now drop off location instead of a pickup location. Because of this the agent must not only learn new paths but it also has to unlearn previous paths.

Due tue to the deterministic nature of random number seeds we cannot supply a “truly random” sample of output. We believe by using a “more random” way of choosing random values we expect to see most experiments using PExploit 1 to perform better on first runs and most experiments using PExploit 2 to perform better on runs where accurate q-tables are already discovered. Overall we discovered Experiment 3 run 2 had the best performance in reaching a terminal state in the fewest amount of steps. Experiment 4 had a high learning rate than Experiment 3 but did not have as high improvement on the second run. This could be a result of the random number seed or the domain of the problem itself.