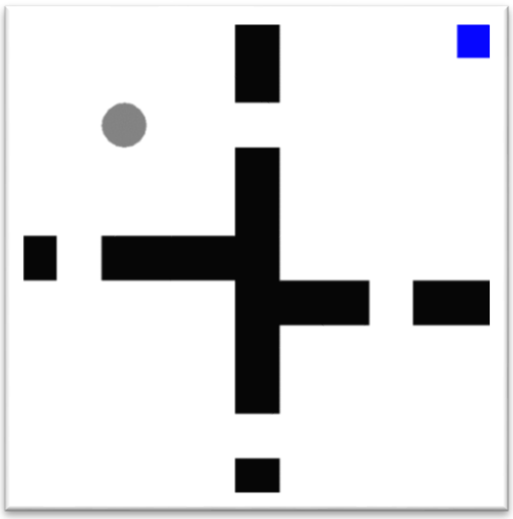
Milind Lingineni CS 4641
Reinforcement Learning Assignment

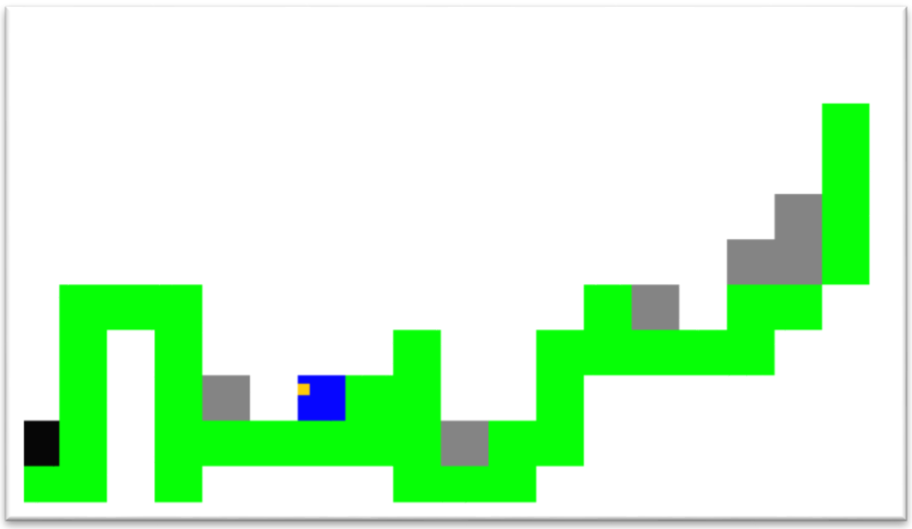
# Reinforcement Learning Report

## MDPs



*GridWorld*: GridWorld is the archetypal reinforcement learning Markov Decision Process (MDP) problem. Within it's simple architecture lies the ability to easily visualize policies and value functions in an easily understandable way. With BURLAP, I used the default 11 x 11 format with four rooms divided by impassible walls and one terminating goal condition in the top right corner. States are represented with integers x and y which represent the location of the agent and also the location of the goal condition. Movement from one state to the next involves taking actions among the cardinal directions: north, east, south, and west. However, movements aren't necessarily deterministic and a stochastic move in a different direction can be implemented. Upon reaching the terminal goal condition, the agent will remain in that state indefinitely without future reward.

*Block Dude*: Block Dude was a flash game first created for the TI-83 calculator. This MDP is significantly more complex than the aforementioned GridWorld problem as there are many more potential states and several more actions to perform. The agent can move left, right, and up if there is only a one block difference in height and can also fall down any height. The block dude can pick up and move any grey block but cannot move of the green blocks. The ultimate goal is to reach the terminal black square which is the terminating goal condition. A poor choice of actions can result in the agent getting stuck. Also, moving alone will not enable the agent to reach the exit; it has to manipulate its environment. BURLAP includes a default level generator but Level 1 and Level 2 yielded exceptions so I decided to use Level 3 which is shown on the next page.



**What makes the MDPs interesting?**

GridWorld is interesting as it's a very simple problem yet it enables you to create intelligible visualizations of the planning or learning process. This contrasts with BlockDude which is too complicated to depict the value function. Another advantage of GridWorld is that it converges to the right answer quickly due to its simple design and limited state space unlike BlockDude which takes significantly longer due to many more states and actions. The ideas in this MDP might also generalize to real life robots like the Roomba or a warehouse robot.

Block Dude is an interesting problem as its state and action spaces are much more complex and can lead to dead ends. It requires much more thought and foresight as it must manipulate the environment to reach its goal. However, with its complexity, the problem is much harder to visualize and understand the learning process. Block Dude is still finite in its number of states and actions though, making it more tractable than a continuous domain. Block Dude is also much more satisfying as an application of reinforcement learning than GridWorld in that it feels like a more challenging game than simply moving to a destination.

**Planners: Value Iteration vs. Policy Iteration**

To compare the two MDPs and planning versus learning, I varied several hyperparameters for the different MDPs and algorithms to change the difficulty and complexity of the problems. To compare value iteration and policy iteration for GridWorld, I altered the probability of a successful transition to the intended state, the reward of the terminal state, and the discount value. For Block Dude, I changed the discount and reward values but not the transition probability as the state transitions are much more complex and any stochasticity might result in the agent inescapably stuck at a dead end.

*GridWorld*

GridWorld is very flexible in that many hyperparameters can be manipulated to analyze how changing several factors impacts the MDP and planners. I'll begin my analysis with the effect of varying the probability of a successful state transition.

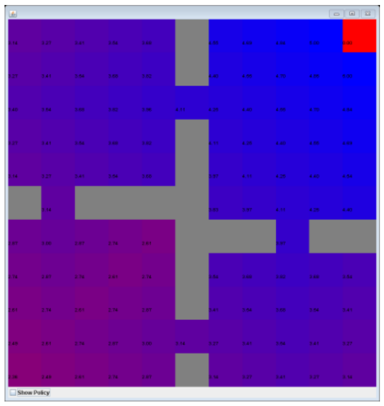
| | **Stochastic Transition** | 0.99 | 0.9 | 0.8 | 0.6 | 0.4 | 0.2 |
|---|---|---|---|---|---|---|---|
| VI | Iterations | 22 | 25 | 30 | 57 | 166 | 349 |
| | Runtime(ms) | 196 | 54 | 51 | 77 | 231 | 568 |
| PI | Iterations of Inner VI | 578 | 448 | 459 | 545 | 703 | 699 |
| | Policy Iterations | 8 | 5 | 5 | 5 | 5 | 4 |
| | Runtime(ms) | 1648 | 1121 | 893 | 694 | 979 | 1291 |

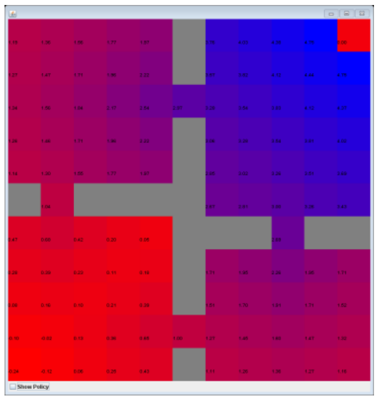
Notice how the number of iterations increases for value iteration and fluctuates somewhat for policy iteration but also somewhat increases as the probability decreases. Oddly though, the runtime seems to decrease initially and then increase afterwards forming a parabolic curve. Value iteration converges much faster than policy iteration both in the number of value iterations and the length of the runtime. Value iteration probably is better at converging in a small state space while policy iteration takes many more iterations as it's not only approximating the value function but a policy directly as well. Policy iteration appears to be better at considering uncertain state transitions though as it's number of iterations appears stable while the number of iterations grew almost exponentially for value iteration as the probability decreased.

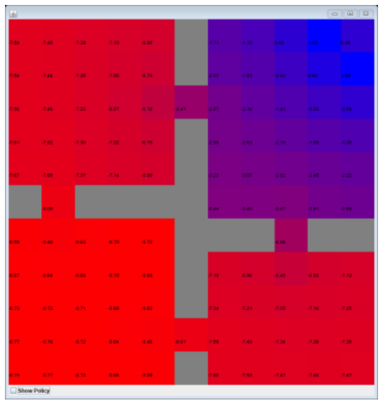
| VI | 0.99 | 0.9 | 0.8 | 0.6 | 0.4 | 0.2 |
|---|---|---|---|---|---|---|
| # of Moves | 21 | 24 | 28 | 57 | 68 | 1103 |

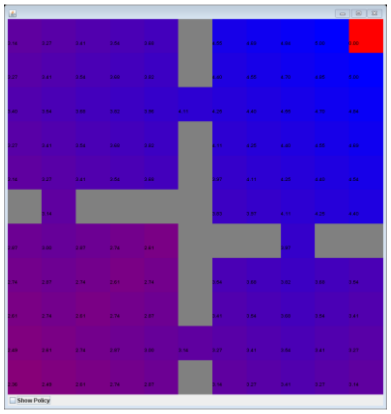
Both value iteration and policy iteration yielded similar value functions though, differing around 0.1 from each other for each grid space. I counted the number of moves needed to reach the terminal condition for value iteration and policy iteration produced similar numbers; however, policy iteration consistently took fewer moves than value iteration. An intriguing result is that lowering the successful transition probability significantly changed the value functions produced. This makes sense as the lower the probability, the less control you have over your next state and the more likely you will be forced to roam endlessly as your actions are not sending you in the right direction.

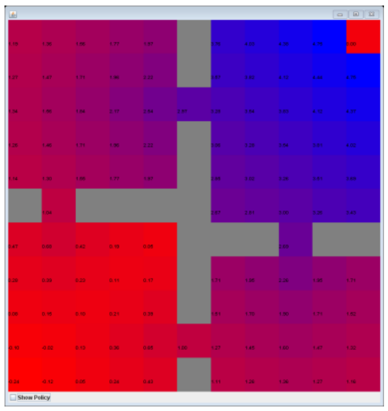
**Stochastic Transition**

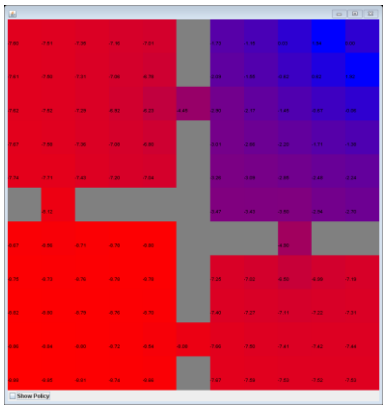

VI 0.99


VI 0.6


VI 0.2


PI 0.99


PI 0.6

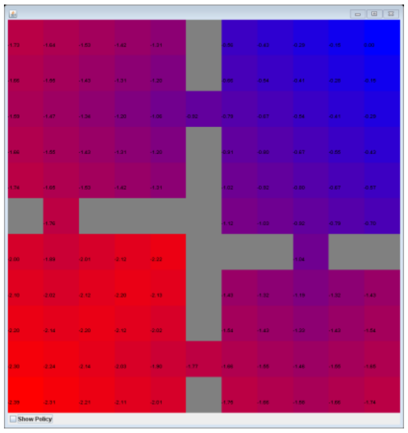

PI 0.2

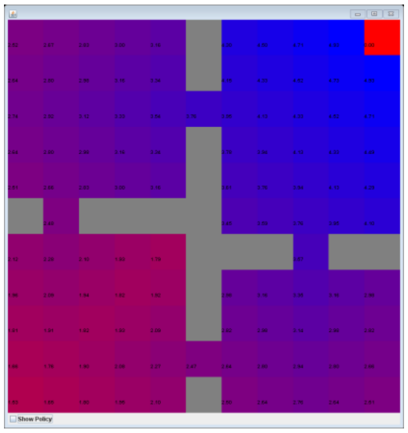
| | **Reward** | -0.1 | 1 | 3 | 5 | 10 | 100 |
|---|---|---|---|---|---|---|---|
| VI | Iterations | 28 | 22 | 25 | 25 | 27 | 30 |
| | Runtime(ms) | 205 | 44 | 44 | 32 | 32 | 44 |
| PI | Iterations of Inner VI | 860 | 716 | 727 | 734 | 750 | 763 |
| | Policy Iterations | 9 | 7 | 7 | 7 | 7 | 6 |
| | Runtime(ms) | 1274 | 805 | 861 | 831 | 840 | 880 |

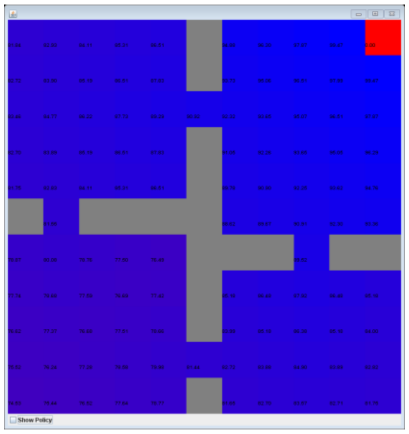
Next, I changed reward, which specifically refers to the reward gained at the goal terminal condition. The reward for each other state remained a constant of -0.1. For different values of reward, the number of iterations and runtime remained relatively consistent except the first one probably due to loading the program into memory. Policy iterations still took significantly longer to run than value iteration to run, probably due to the nature of the

GridWorld MDP. I’ve read that policy iteration should converge faster, but maybe this doesn’t hold because GridWorld is so simple that value iteration converges faster.

**Reward**


VI/PI -0.1


VI/PI 5.0

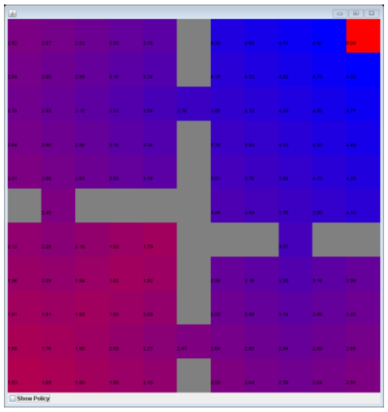

VI/PI 100.0

For all the different values of reward used, i.e. {-0.1, 1, 3, 5, 10, 100}, both value iteration and policy iteration generated the same value functions and policies for the corresponding reward. The values in all the graphs above are exactly identical in their floating-point values. Upon running each different reward and planner, each agent nearly followed the optimal 21 move path to the terminal condition but wavered back and forth occasionally due to a default successful transition probability of 0.8. When the goal reward is -0.1, which is equal to the reward of every other state, the areas near the terminating state have higher values as they know that a termination would end the accumulation of negative reward. In stark contrast, a goal reward of 100 results in every state having a high value as they include the goal reward, but the agent can still make a policy as the states closer to the end have higher values due to the default discount of 0.99. A goal reward of 5.0 appears to be a blend of the value functions of -0.1 and 100.

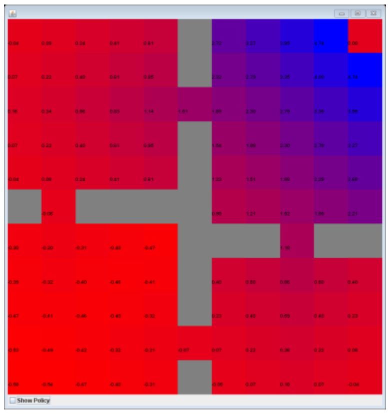
| | **Discount** | 0.99 | 0.95 | 0.9 | 0.8 | 0.6 |
|---|---|---|---|---|---|---|
| VI | Iterations | 22 | 21 | 21 | 21 | 14 |
| | Runtime(ms) | 33 | 34 | 33 | 45 | 24 |
| PI | Iterations of Inner VI | 578 | 290 | 143 | 68 | 34 |
| | Policy Iterations | 8 | 7 | 6 | 7 | 6 |
| | Runtime(ms) | 758 | 597 | 258 | 135 | 77 |

Lastly, I changed the discount value which determines the importance assigned to future events. For value iteration, the number of iterations remained consistent and was ultimately able to find the optimal policy for all but 0.6. This was probably because having such a low discount number made the far-away goal reward unappealing. For policy iteration, the number of value iterations done and the runtime decreased consistently with decreasing discount value.

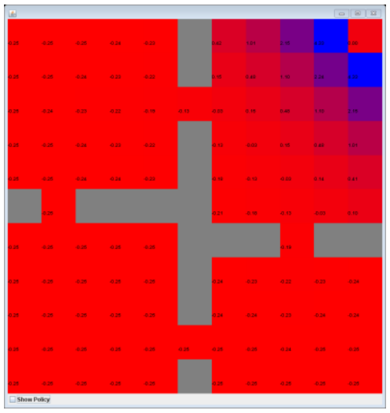
**Discount**



VI/PI 0.99



VI/PI 0.90



VI/PI 0.60

Changing discount, represented as gamma, can radically alter the value function as seen in the images above. As had happened with reward, both policy iteration and value iteration yielded the same value functions and visualizations. Having a discount value near 0 means that we’re focusing mostly on the reward of the neighboring states. As demonstrated by a gamma of 0.6, only states near the goal condition have positive utilities while the rest have negative utilities. When the discount value is near 1, we do the opposite and focus the combined rewards farther into the future. The gamma of 0.99 is the antithesis of the 0.60 visualization, as all the states have positive values which account for the goal reward, despite the terminating state being far away from many of the other states. Even going from 0.99 to 0.90 was enough to make half the board red with negative values because the discount is exponentiated.

## *Block Dude*

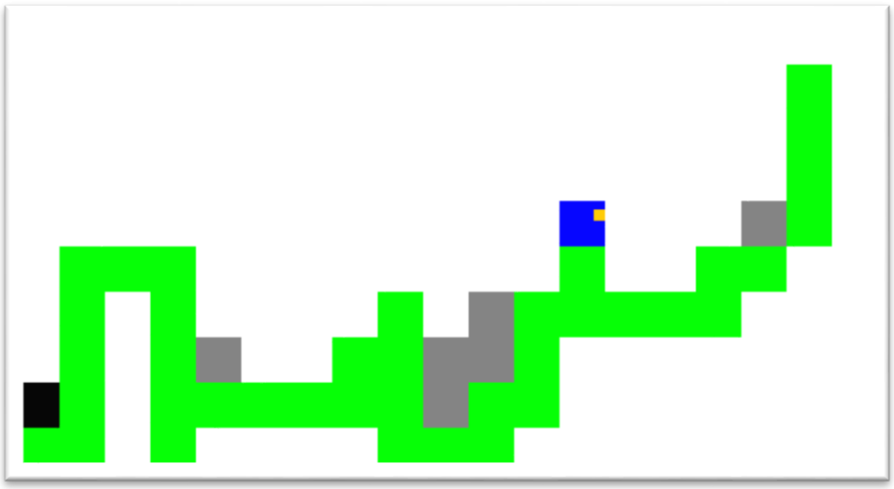
Block Dude has significantly more possible states than GridWorld and is more representative of an actual video game as it was played on Texas Instrument calculators. From the output of BURLAP, we can see that there are 14200 possible states for the Block Dude level I’m using while the GridWorld only has a paltry 104 states. This makes Block Dude significantly more challenging to solve, resulting in much larger numbers of iterations and longer runtimes than seen in GridWorld.

| | **Discount** | 0.99 | 0.95 | 0.9 | 0.8 | 0.6 |
|---|---|---|---|---|---|---|
| VI | Iterations | 688 | 135 | 66 | 31 | 14 |
| | Runtime(ms) | 168442 | 31287 | 15109 | 10266 | 4276 |
| PI | Iterations of Inner VI | 2767 | 967 | 474 | 166 | 35 |
| | Policy Iterations | 47 | 32 | 28 | 21 | 10 |
| | Runtime(ms) | 874593 | 287425 | 141302 | 57895 | 11904 |

As depicting the value function is more difficult for Block Dude than GridWorld, we'll need to consider runtime and iterations instead. Both value iteration and policy iteration decreased in their respective iterations and runtimes as discount decreased. This differs from what occurred with GridWorld as value iteration seemed to take similar iterations and runtimes for different discount values. With Block Dude instead, both iterations and runtime decrease drastically as discount increases, probably due to the complexity of the Block Dude MDP. At discount values of 0.8 and 0.6, value and policy iterations would occasionally generate policies that did not reach the goal condition. This might be because the future reward of the end goal was reduced to nothing by the exponentially decreasing future discount. Discount values of 0.99, 0.95, and 0.9 consistently produced the optimal solution though but took significantly longer to run as it had to consider rewards far into the future.

Changing the goal reward value did not appear to affect the value iteration or policy iteration planners. This might be due to the default state reward of -0.1 so the agent will always be incentivized to reach the terminating state regardless of reward. Each value iteration and policy iteration took the same number iterations and length of runtime for each reward value, so ultimately changing the reward doesn't matter for this MDP. They all arrived at the same solution too. These results are similar to those of the GridWorld MDP as the reward would change the value function but not the path to the goal. An agent almost reaching the goal state can be seen on the next page.

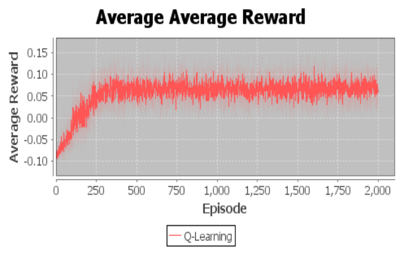
| | **Reward** | -0.1 | 1 | 3 | 5 | 10 | 100 |
|---|---|---|---|---|---|---|---|
| VI | Iterations | 688 | 688 | 688 | 688 | 688 | 688 |
| | Runtime(ms) | 154251 | 151661 | 150931 | 159716 | 161058 | 191023 |
| PI | Iterations of Inner VI | 281 | 281 | 281 | 281 | 281 | 281 |
| | Policy Iterations | 47 | 47 | 47 | 47 | 47 | 47 |
| | Runtime(ms) | 1042525 | 7038393 | 1291000 | 822794 | 980422 | 974074 |



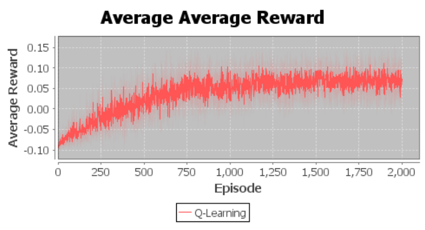
## Learner: Q-Learning

I chose Q-Learning to solve my two MDPs which differs from the planners as it doesn't require the transition and reward functions given by a model. Instead, it learns the policy and Q function by interacting with the environment using pure transitions and is a model-free approach. I chose number of steps per episode and average reward per episode to analyze the progress of the agent. The number of steps enables us to determine the time until convergence while average reward tells us how the reward gained improves over more episodes and where it converges to. These metrics will help to juxtapose the two problems.

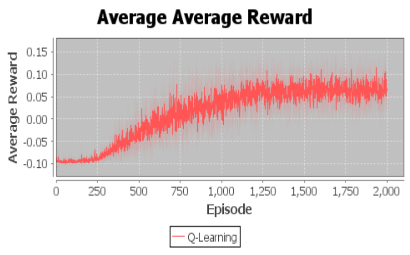
### *GridWorld*



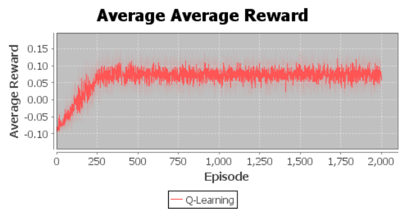
qInit 0.30



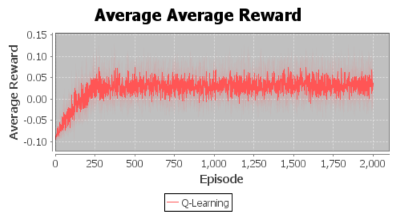
qInit 5.0



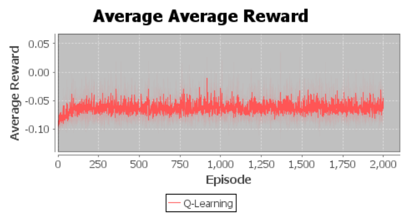
qInit 30.0



epsilon 0.05



epsilon 0.30



epsilon 0.80

For GridWorld, I began Q-Learning by altering the qInit values. These are the initial values that are assigned to any unexperienced state-action pair. The Q-values of {0.3, 0.5, 1.0} all yielded similar average reward curves as seen for qInit of 0.30. When qInit was changed to larger values though, the agent started to become slower to converge to the optimal average reward. These results are significant as it means that initializing to larger values means taking longer periods of time to replace the placeholder values with the actual values. This slower convergence is caused by a constant epsilon so increasing the qInit value increases the value used for exploitation and it takes longer for exploration to correct the incorrect exploitation assumptions.

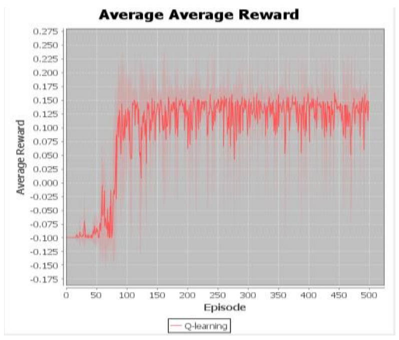
Changing the epsilon values also produced some interesting results. As epsilon represents the probability of choosing a random action and is therefore exploration, increasing epsilon appears to decrease the average reward. The increased probability of exploration leads to more time visiting states and less time exploiting the information gained to converge to the optimal policy. For each increase in epsilon for the values {0.05, 0.1, 0.3, 0.5, 0.8} consistently resulted in a lower final average reward with increasing epsilon. Higher values of qInit seemed to lead to longer runtime. This same trend held with higher epsilon values taking longer runtimes as well. As noted before, the first value used always seems to take longer than normal which is probably due to loading the program and not the fault of the value itself.

**GW**

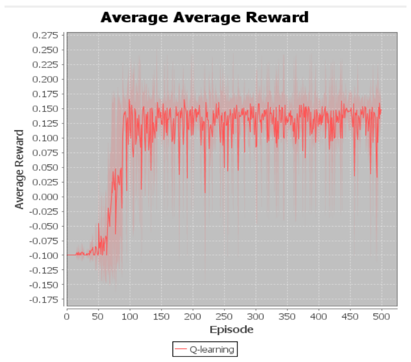
| qInit | 0.3 | 0.5 | 1 | 5 | 30 |
|---|---|---|---|---|---|
| Runtime(ms) | 1693 | 1138 | 1256 | 1390 | 6806 |
| epsilon | 0.05 | 0.1 | 0.3 | 0.5 | 0.8 |
| Runtime(ms) | 2105 | 1323 | 1677 | 1769 | 4319 |

*Block Dude*

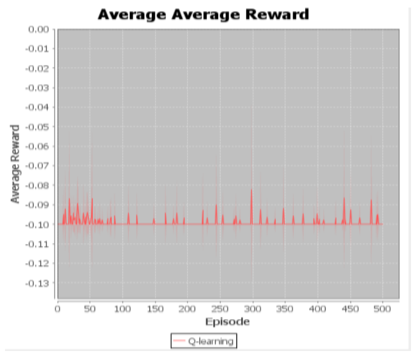
For the Block Dude MDP, I changed the same two parameters as I'd done for GridWorld: the qInit values and epsilon. Many of the same trends occurred for Block Dude as had happened for GridWorld. First, I'll look at average reward with different qInit values. The values {0.3, 0.5, 1.0, 5.0} produced the same average reward graphs; however, a qInit value of 30 caused the average reward to flatline drastically to zero. This means that the Q-Learning algorithm failed to converge to the optimal policy. This result was probably because a high initial Q-value forces the algorithm to take longer to correct the poorly initialized value and has



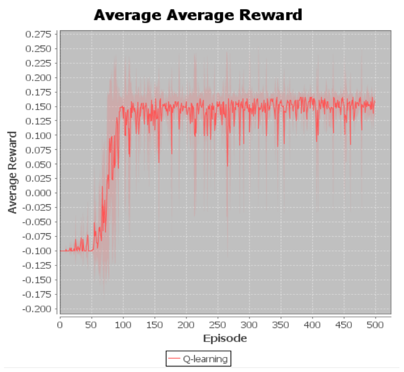
qInit 0.30



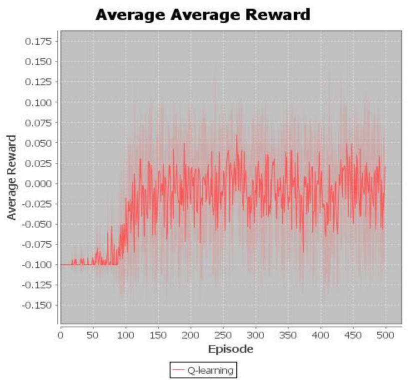
qInit 5.0



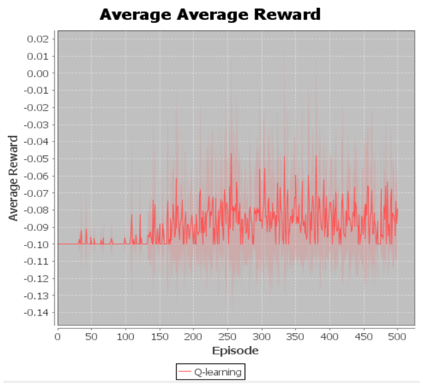
qInit 30.0



epsilon 0.05



epsilon 0.30



epsilon 0.80

to spend more time exploring to correct it to the right value. I believe that given more episodes, the graph would eventually look like the qInit 30.0 graph that the GridWorld MDP produced.

Increasing the epsilon value resulted in worse average reward graphs. This exactly follows the movement that occurred with GridWorld. Due to the exploration-exploitation tradeoff, increasing the epsilon value resulted in greater time exploring and less time exploiting the learned Q-function. This results in lower reward as can be seen by the immense difference between an epsilon of 0.05 and 0.80. In fact, the average reward stays negative for values of 0.30, 0.50, and 0.80. The same trends hold for runtime for both qInit and epsilon as had occurred with GridWorld. The increased time needed for exploration caused by high initial Q-values and high epsilon values resulted in the longer runtimes.

**BD**

| | | | | | |
|---|---|---|---|---|---|
| qInit | 0.3 | 0.5 | 1 | 5 | 30 |
| Runtime(ms) | 772 | 579 | 594 | 673 | 2836 |
| epsilon | 0.05 | 0.1 | 0.3 | 0.5 | 0.8 |
| Runtime(ms) | 510 | 635 | 908 | 1317 | 2493 |