Shannon Ke
CS 4641
Reinforcement Learning Report

Markov Decision Processing Problems

Grid World

The grid world is a simple problem dealing with an agent learning an optimal path through a maze to get from point A to point B. This type of problem is especially applicable in the world of robotics or even that of games. For example, arcade games such as Pacman that can be solved with game AI are essentially just grid world mazes, where each piece of food in the maze is a state with high positive reward, and each ghost's surrounding areas carrying a high negative reward. In the field of robotics, the real world can be constructed as a grid world, allowing robots to navigate through buildings with the goal location having the highest reward. The grid world problem was also used because of its relatively small state space, with the small 5x5 grid world used having a state space of 17, and the large 11x11 grid world used having a state space of 74.

Both grid worlds have penalties for every state of -1 except the goal state, which gives a reward of +100. However, there are also states with penalties of -100 included in the hard grid world, which will be explained in more detail in later sections.

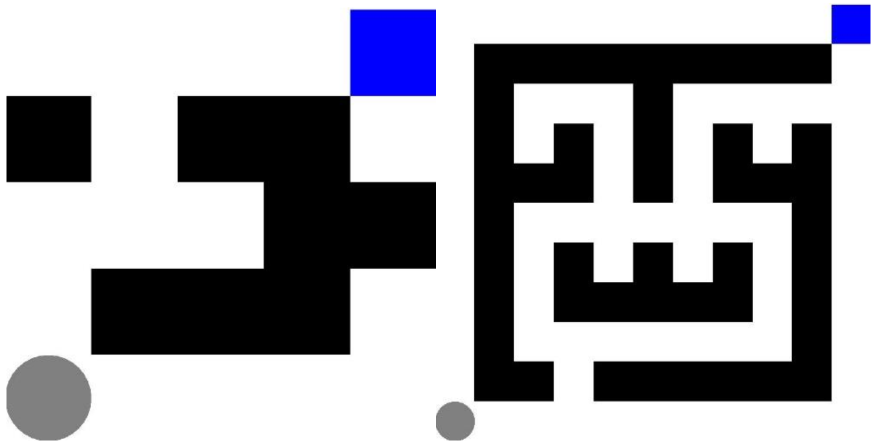

Figure 1(a and b). Diagrams detailing the easy and hard grid worlds used in this project. (a) Initial runs on the easy grid world did not include unreachable states and dead ends, but in doing so, I could not find any difference between value and policy iteration on the world. Thus, a wall was placed on the right side of the map, creating a dead end and a state that theoretically can never be reached. However, given the natures of value and policy iteration, they should now act slightly differently. This will be analyzed in later sections. (b) The large grid world is interesting since I left the two shortest paths open (the ones bordering the edges) but gave them high negative rewards. Also, I wanted the grid world to look like a face as much as possible since I thought it looked cute. Note in the graphs later how the utility mapping either looks like a cute bug or a laughing face with large lips.

Block Dude

Blockdude originated as a Texas Instruments calculator game, with the player being a man who is able to move blocks in order to get to an exit. The player can fall any distance but can only go up by a distance of one block. Blockdude is an interesting problem because of its large state space, with the hard level having 14199 possible states and the easy level having 840. Blockdude also represents a more complex problem than grid world, since along with figuring out how to get from point A to point B, the agent must also figure out how to solve the puzzle of getting over tall walls by moving blocks to their correct positions. Since the nature of Blockdude involves more critical thinking, running reinforcement learning algorithms on it also represents how such algorithms can also perform in other more complex real world situations.

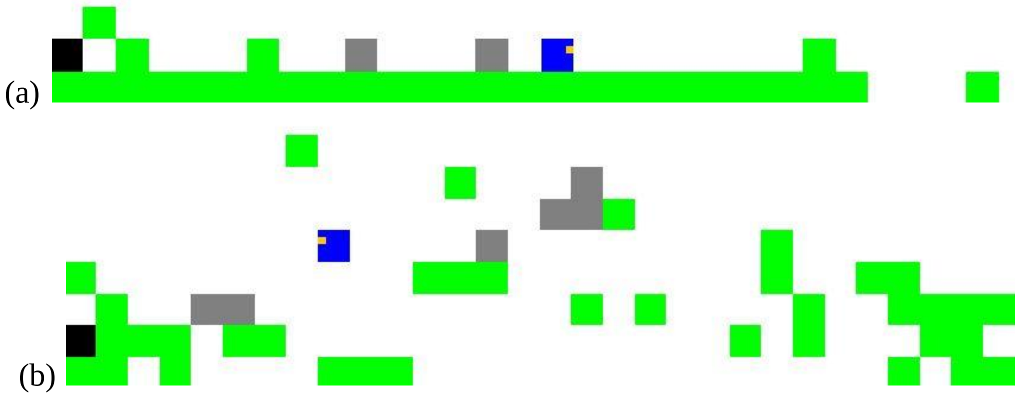

Figure 2(a and b). The two levels of Blockdude, with (a) being easy and (b) being hard. The actual visualization was accompanied with a step by step breakdown of Blockdude's optimal policy, which can be seen in the GUI by running the BlockDudePI.java and BlockDudeVI.java files. I am not certain how Blockdude manages to make it to the exit door (the black square) in the harder level (b), but I am assuming that the graphic itself accidentally included an extra green wall block above the door and that it was not actually evaluated as an obstacle.

General Methodology

Using BURLAP, the recommended machine learning library and some online GitHub repositories which are documented in the README, I was able to run two grid worlds of different difficulties and two Blockdude scenarios of different difficulties with value and policy iteration. In the util files, the specific path of which is also in the README, I could change stochasticity values and gamma values for an interesting analysis. I ran all four scenarios with constant gamma values at 0.99, meaning each algorithm chooses actions that is more likely to optimize a future goal, while changing stochasticity values, or the probability that the agent will move in a certain direction at a certain state. Finally, I chose Q-learning as an additional reinforcement learning algorithm to assess grid world and Blockdude on and compared its performance with value and policy iteration. Q-learning is an interesting reinforcement learning algorithm since it is an online learning algorithm, whereas value and policy iterations are offline.

Value and Policy Iterations - Grid World

Easy Grid World

The easy grid world is a 5x5 world with a state space of 17. I first held gamma values at a constant 0.99, meaning the agent will find an optimal path while prioritizing future rewards. Through the trials, I changed the stochastic values, varying them from 0.25 to 0.95 (a value close to 1 but not quite). The stochastic value is a value that determines the probability that the agent will choose a direction that is not the direction it originally was going to choose. For example, a stochasticity value of 0.8 would mean the agent has an 80% chance of going in the direction it planned to go and 20/3% chance to go in either of the other 3 directions.

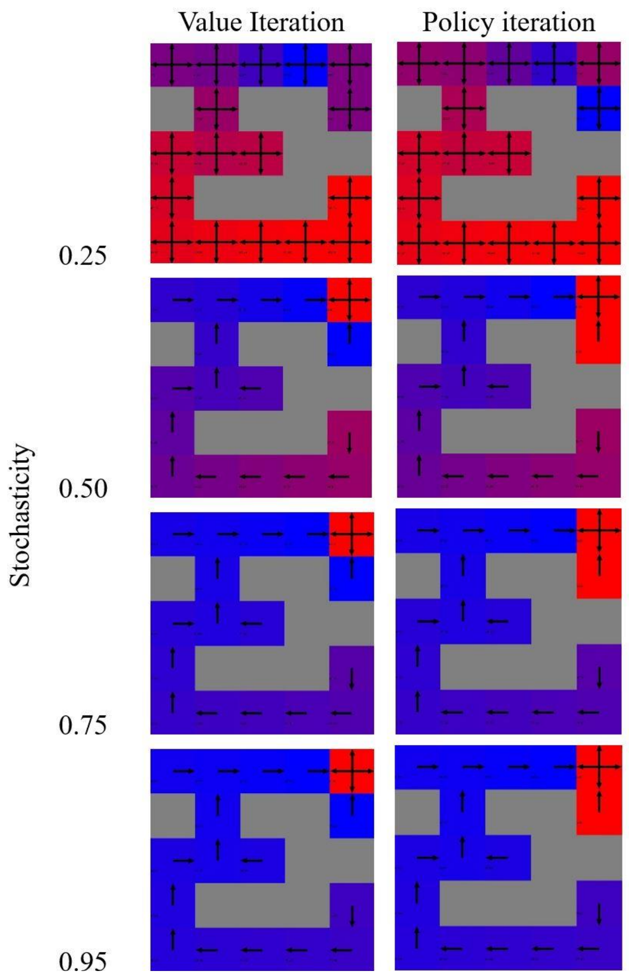

Figure 3. The easy grid world run with value and policy iteration with varying stochasticities. The arrows show optimal policy, although they are quite difficult to see in this chart. The main point of focus for those is that the optimal policies are the same for both value and policy iterations.

Since having a stochastic value as low as 25% is equivalent to randomly choosing a direction to move in, clearly it is not the best choice. The optimal policies, represented by black arrows, for both value and policy iterations show in all four directions for every state as a result. Clearly, having a stochastic value that is too low is not optimal for finding a good policy.

With increasing amounts of stochastic value, we can observe increasing utility for each state (represented by a more blue color) throughout the grid world. Utility is low for low stochastic values since the agent will more likely move in a direction it does not think is optimal. As such, an optimal policy cannot be converged upon since at every state, the agent is not able to pinpoint and go in the direction of the optimal policy.

Predictably, with less noise, the agent is able to find the goal through the optimal policy with less uncertainty. And while value and policy iteration both returned the same exact optimal policy, note the one state past the goal state (top right square) that is colored differently between the two algorithms. This is due to the natures of value and policy iteration. While the value function in value iteration looks to update the Q value for every state separately from finding an optimal policy, policy iteration focuses on finding the optimal policy from the get-go. Thus, it makes sense for value iteration to see the dead-end square as a great spot since it is very close to a high reward while policy iteration understands that the agent traveling under the optimal policy could never make it to that state. Remember that value iteration can converge on an optimal policy before the value function converges as well.

Hard Grid World

Given these facts, it makes sense that while each iteration of value iteration is computationally fast, it takes many iterations to converge, whereas policy iteration starts by initially creating a random policy and then optimizing that instead, leading to fewer total iterations but more computational complexity. Let's see how these two algorithms perform on the large grid world.

The large grid world was set up so that the path to the goal (top right square) was the shortest along the outer edges and longer by traversing the inside of the maze. However, states with penalties of -100 were placed on the shorter paths. As expected from the results of the small grid world, both algorithms were able to find the optimal policies and learn to avoid the outer edges with high negative rewards. Again, with the low value of 25%, an optimal policy could not be found since the agent had an equal probability of landing in any direction given a state. However, low utility is still detected where penalties were great. With increasing stochastic values, I observed a higher utility overall and early detection of penalizing actions, not unlike how the algorithms performed on the small grid world. Both value and policy iteration again converged onto the same optimal policy, with little to no noticeable difference between the two. Unlike the small grid world, I was unable to run the large grid world with a stochastic value of 0.95 due to hardware limitations. I predict that it would have behaved similarly to 0.75 but with less uncertainty. However, it is beneficial to have some amount of stochasticity in an environment, since real applications of MDPs rarely deal with completely deterministic environments. I will now summarize iteration, runtime, and computational complexity of each algorithm on each grid world.

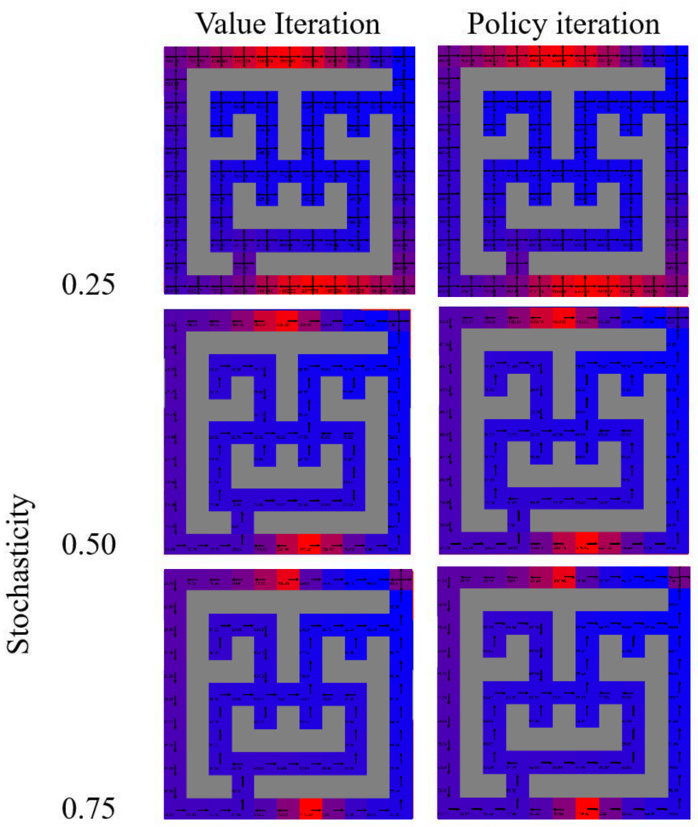

Figure 4. The hard grid world run with value and policy iteration with varying stochasticities. The arrows show optimal policy, although again, they are quite difficult to see in this chart. The main point of focus for those is that the optimal policies are the same for both value and policy iterations.

Grid World Summaries

To summarize value and policy iterations over both grid worlds, I kept the discount factor at 0.99 and the stochastic value at 0.75. Based on the aggregate analysis of the two algorithms, I was able to plot visualizations of number of steps/actions taken, number of milliseconds required, and amount of reward gained for each over 100 iterations.

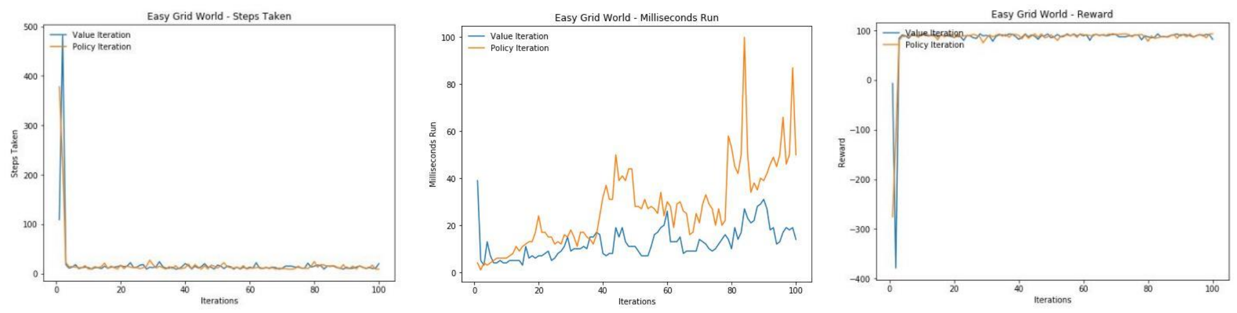

Figure 5. (a) The number of actions required to converge on the optimal policy plotted with respect to number of iterations. Though both were able to converge quickly onto an optimal number of actions, notice how value iteration peaks above policy iteration by more than 100 steps. (b) The milliseconds required for each algorithm to run plotted with respect to number of iterations. Notice how the overall trend is for policy iteration to take longer than value iteration. (c) Total reward plotted with respect to number of iterations. Both iterations are able to quickly converge onto the optimal policy, so reward is also quickly maximized. Notice how (c) is the inverse image of (a).

The graphs in *Figure 5* summarize well the observations I made in the previous section. *Figure 5a* shows that value iteration required more steps initially than policy iteration. This makes sense because value iteration focuses on optimizing the Q value of each state and thus requires many computations for each state. This is reflected by the initially lower reward of value iteration in *Figure 5c.* However, each Bellman update in value iteration has lower computational complexity than that of policy iteration since policy iteration involves updating an entire policy, while value iteration only involves updating states to find an optimal policy. Thus, value iteration spent lower overall time to run as displayed by *Figure 5b.* We can assume that results will be similar for the hard grid world.

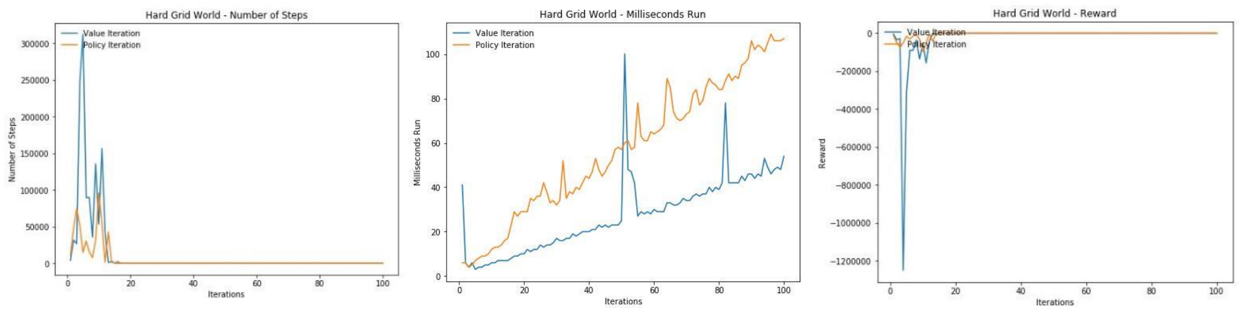

Figure 6. (a) The number of actions required to converge on the optimal policy plotted with respect to number of iterations. Though both were able to converge quickly onto an optimal number of actions, notice how value iteration peaks above policy iteration by more than a few hundred thousand steps. (b) The milliseconds required for each algorithm to run plotted with respect to number of iterations. (c) Total reward plotted with respect to number of iterations. Both iterations are able to quickly converge onto the optimal policy, so reward is also quickly maximized. Penalty values are exponentially larger than in the easy grid world, due to increased number of states as well as the existence of multiple states with very large penalties. Thus, it is a bit hard to see, but reward does converge to around 60-70.

Value and Policy Iterations – BlockDude

The easy level of BlockDude contains 841 states, with the optimal policy taking 19 steps to the exit. Total reward for both value and policy iterations was 3.2, although, as predicted by the grid world analysis, the runtimes were vastly different. The hard level of BlockDude contains a whopping 14200 states and required 94 steps for the optimal policy. It had an overall reward of -4.3. A further summary of results is diagramed in the table below.

	Easy VI	**Hard VI**	**Easy PI**	**Hard PI**
Number of States	841	14200	841	14200
Runtime	8778 ms	88668 ms	17817 ms	1833686 ms
Steps Taken	19	94	19	94
Iterations	230	230	20	52
Total Reward	3.1999999	-4.299999	3.199999	-4.299999

Figure 7. This table summarizes results from running value iteration (VI) and policy iteration (PI) on BlockDude. Values that are most notable are runtime and iterations.

BlockDude was a useful problem to analyze differences in number of iterations needed to converge and the drastic difference in runtime for each algorithm. Runtime for policy iteration on the easy BlockDude map takes more than twice as long to converge with less than a tenth of the number of iterations. Similar behavior is exhibited in the hard BlockDude level. This demonstrates my earlier finding, which is that while value iteration has lower overall computational complexity, it requires more passes or iterations to converge on an optimal policy. In contrast, policy iteration required only 20 and 52 passes for the easy and hard levels respectively but took much longer to converge, with the hard level taking 1833686 milliseconds. Also as expected, both value and policy iteration were able to converge onto the same optimal policy.

Reinforcement Learning – Q-learning

Q-learning was the algorithm chosen to test both the grid worlds and BlockDude on, since in contrast from value and policy iteration, it is an online learning algorithm. This means that state transition and reward models are unknown to the agent at first, and the agent can only observe the current environment state. The agent must constantly be actively learning about its environment at each state. This is an interesting and relevant reinforcement learning algorithm since the anonymity of the world surrounding the agent is more akin to real world situations, where a robot may not understand its surroundings prior to entering its environment.

Grid World

Q-learning was run on both the easy and hard grid worlds. The results have been displayed below.

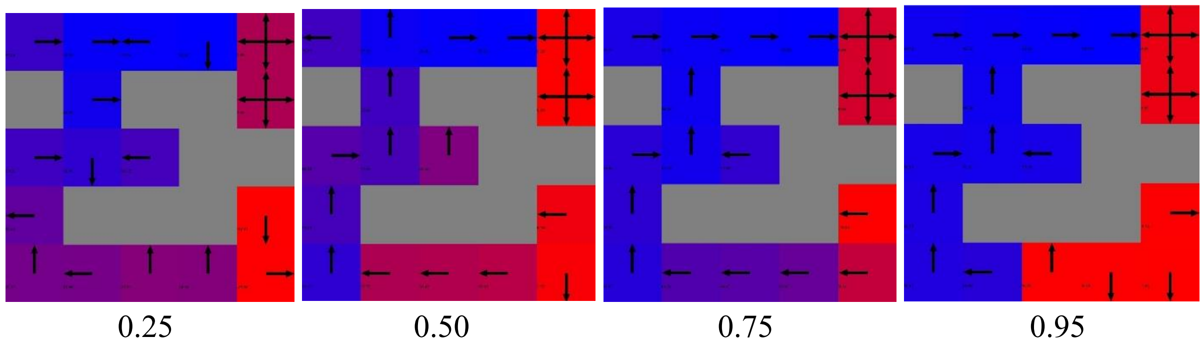

Figure 8. Q-learning was run on the easy grid world for stochastic values of 0.25, 0.5, 0.75, and 0.95, just like for value and policy iteration. Q-learning produces interesting policies from the start, even producing a policy with stochastic values as low as 0.25, which is unlike value and policy iteration.

Already, we can see that Q-learning behaves quite differently from value and policy iteration. Like policy iteration, it sees that the dead end after the goal state is unviable, but it also has some interesting behaviors. While both value and policy iterations were unable to come up with policies with stochastic values at 0.25, Q-learning still attempts to make its (albeit incorrect) optimal policy. This is because Q-learning does not rely on transition probabilities to come up with an optimal policy since it has no knowledge of a transition model. Q-learning chooses a next action to take by making sure it maximizes the next state's Q-value instead of relying on a current policy. Thus, Q-learning is known as an off-policy reinforcement learning algorithm. So although Q-learning was unable to create the optimal policy with low stochastic values, it was still able to create a policy regardless.

Another interesting point to notice is that with high stochastic values, Q-learning was able to detect that the area on the bottom right led to a dead end and gave those states extremely low utilities, something policy and value iteration did not do. This is because each iteration of Q-learning puts the agent in a black box, meaning the agent cannot see the full transition model. It simply knows that in taking a certain action at a certain state, a sure way to not maximize the next Q-value is to head into the dead end in the right corner.

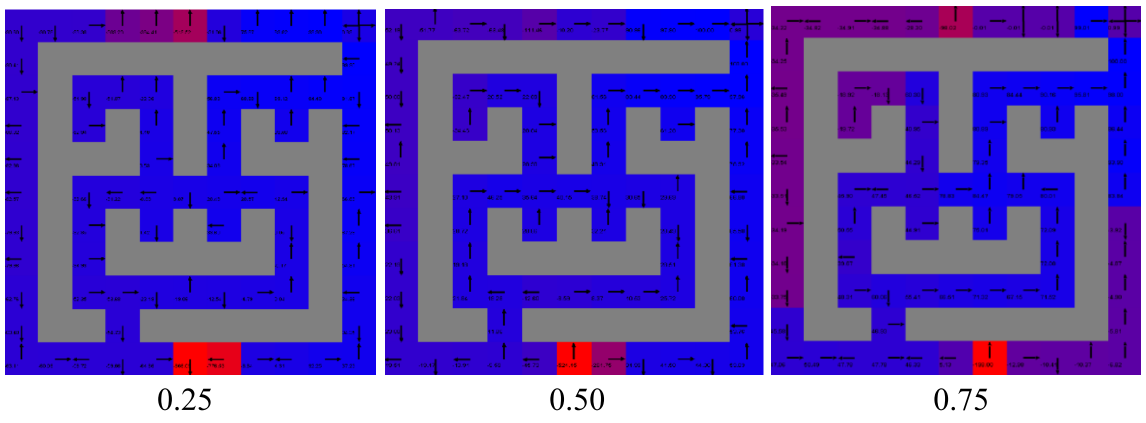

Figure 9. Q-learning was run on the hard grid world for stochastic values of 0.25, 0.5, and 0.75, just like for value and policy iteration. Given the complexity of running the algorithm with a stochastic value of 0.95, I only ran the algorithm on the three stochastic values displayed.

As expected, Q-learning behaves similarly on the large grid world as the small grid world. It attempts to create an optimal policy with low stochastic values, showing that those values do not affect Q-learning's ability to create a policy. Q-learning is able to see and localize early on the cost of the -100 penalty states. Also, the algorithm puts a lower utility on dead ends, further solidifying the idea that it updates its Q function based on maximizing the next state's Q-value.

Note that Q-learning does not actually converge on the optimal policy. This can be seen by the arrows not all pointing in the correct directions towards the exits, even with high stochastic values. This is most likely due to insufficient number of iterations, since given infinite exploration time, Q-learning should always converge on the optimal policy.

In comparison to value and policy iterations, Q-learning did not experience overwhelming tips and spikes in data and was more stable with its results across 100 iterations. This can be attributed to Q-learning's off-policy nature, making it spend a bit more time to converge than the other learning algorithms given an absence of the transition model. Overall, number of steps taken and reward are less spread out but do not converge as quickly as value and policy iteration, as shown in *Figure10a* and *Figure10c.* However, note that overall runtime is faster than that of value iteration and much faster than that of policy iteration. It makes sense that it is closer to value iteration considering that value iteration and Q-learning are closely related. This implies that even though the transition model is unknown, Q-learning is still a very efficient algorithm that, if given enough iterations, can also reliably converge on the optimal policy.

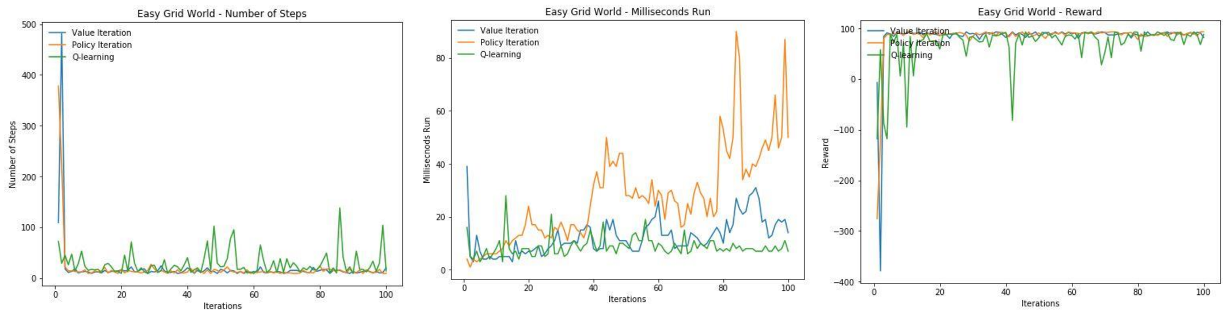

Figure 10. Q-learning has been superimposed onto the graphs from *Figure5*, the easy grid world analysis. Overall trends here indicate that although Q-learning does not immediately converge onto the optimal policy, it also takes less time and starts off already relatively close to its goal.

BlockDude

Due to computational complexity and hardware limitations, Q-learning for BlockDude was run only on the easy level. The algorithm was run while varying one of three initial values. Epsilon was held at 0.05, learning rate at 0.5, and initial Q at 0.1. Each experiment varied just one of those parameters. Results are shown colorfully below.

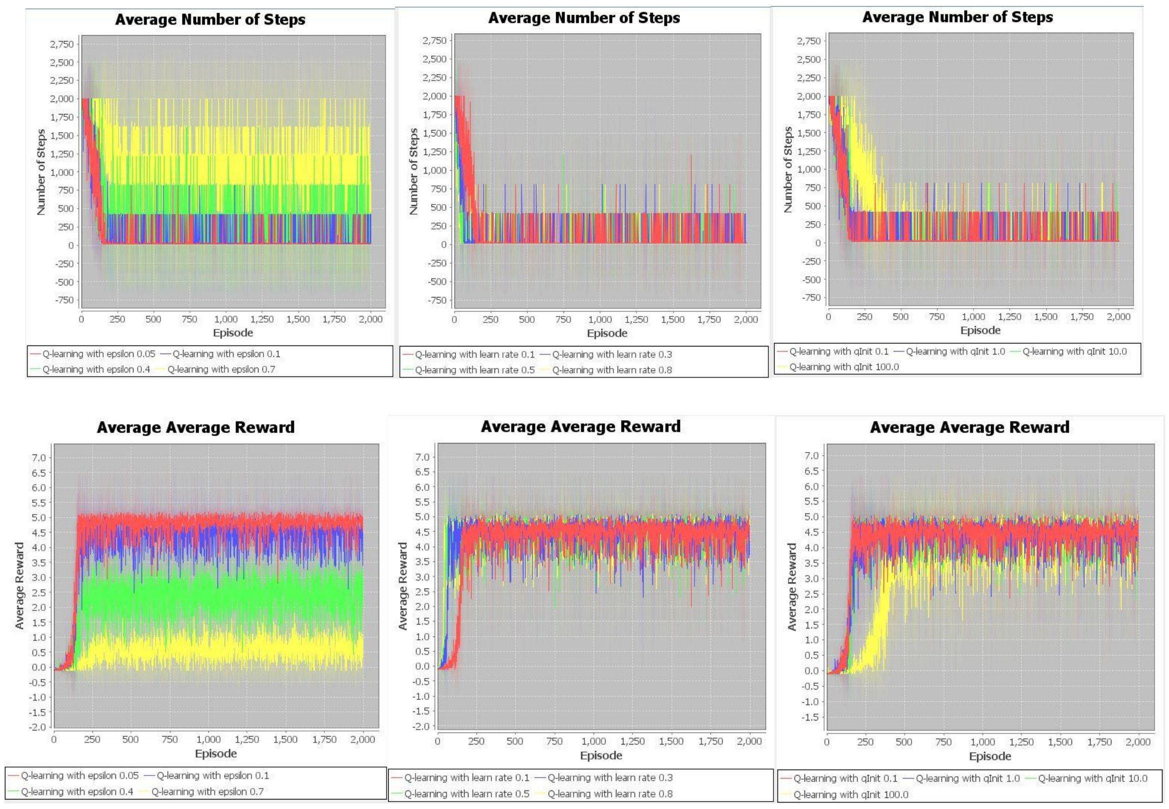

Figure 11. Average number of steps and reward for Q-learning run on the easy BlockDude level. The first column varies epsilon values, the second column varies learning rate, and the third column varies initial Q. Overall, Q-learning performs better with low epsilon, about the same for all learning rates, with higher learning rates being more optimal, and with a low initial Q value.

This experiment introduces the idea of exploration vs. exploitation, a problem that Q-learning has when trying to select an action based on learnt Q values versus trying to select an action randomly in hopes for a better reward. An approach to solving this problem is known as the epsilon-greedy approach, with a smaller value of epsilon meaning a higher likelihood of choosing a random action and exploring. This is interesting behavior for BlockDude, since the state space for the easy level is rather small and should not require as much exploring. From the second column dealing with varying learning rate in *Figure 11,* we can see that a higher learning rate caused faster convergence, and in the third column, that high initial Q values slowed down convergence. Lower initial Q values mean that Q(a, s) for all other states will be higher than the initial value, making returning to already visited states more attractive (aka exploitation and less exploration). For the small space of the easy BlockDude level and the nature of the game (to go back and forth between states while carrying a block), it makes sense that lower initial Q values work better.

Comparatively, BlockDude's Q-learning mirrored the behavior of the grid world's Q-learning trends in terms of run time. Also like grid world's Q-learning, BlockDude's Q-learning does not converge completely and instead spikes around an average. However, it does converge quickly and has about the same reward as value and policy iteration. It is also worth noting that BlockDude is not merely a maze that requires an agent to go from point A to point B, but an actual game that requires more learning and adaptation for the algorithm, which most likely explains the high spikes of steps taken and low reward at the start of the algorithms.

Conclusion

After running value and policy iteration and Q-learning on two grid worlds and two levels of BlockDude, there are some clear trends that have surfaced. Value iteration and policy iteration are both offline algorithms, or on-policy, meaning they have access to transition and reward models. Thus, they converge in less iterations than Q-learning, which does not have access to those models. Both value and policy iteration have their pros and cons. Value iteration only looks to optimize the Q values for each state using the Bellman equation, which could lead to the creation of an optimal policy before fully converging. With smaller updates being performed every iteration, value iteration is not as computationally complex as policy iteration, making it faster. Policy iteration, on the other hand, looks to optimize a policy by using the Bellman equation to create a new policy with every iteration. Thus, it converges in fewer iterations but is more computationally complex than value iteration, meaning it takes longer. Finally, Q-learning is more similar to value iteration, in which it looks to optimize Q values in every iteration, but since it does not have access to a reward or transition model, Q-learning only has to worry about updating its utility for the next action state, making the computations even simpler than value iteration. Hence, Q-learning works faster than value iteration with respect to number of iterations. However, it requires more iterations to converge.

Parameters that can be varied for value and policy iteration include stochastic values (which was done) and gamma values (which was not done). An interesting experiment would be to vary gamma values, which control how far in the future the algorithms look to optimize values. In other words, having higher gamma values mean that the algorithms will look to optimize rewards in distant states. It made sense to me to leave gamma values high, since in both grid world and BlockDude, we are looking for just one goal state to work towards and there were not many tricks or pitfalls present in the scenarios. However, it would still be interesting to analyze performance differences while varying this metric.

Parameters that I did vary for Q-learning include initial epsilon, learning rate, and initial Q values. Epsilon and initial Q values control the algorithm's tendency to explore vs. exploit and learning rate controls how fast the algorithm converges on a policy. Q-learning was an interesting reinforcement learning algorithm to use since it tackles a different problem using similar methods as value and policy iteration. While value and policy iteration are useful and reliable algorithms, most of the time in the real world, transition and reward models will be unknown, just like how in the real world, environments are normally stochastic and not deterministic. Q-learning is able to tackle problems with many unknowns and still find an optimal policy, making it a versatile and applicable reinforcement learning algorithm.