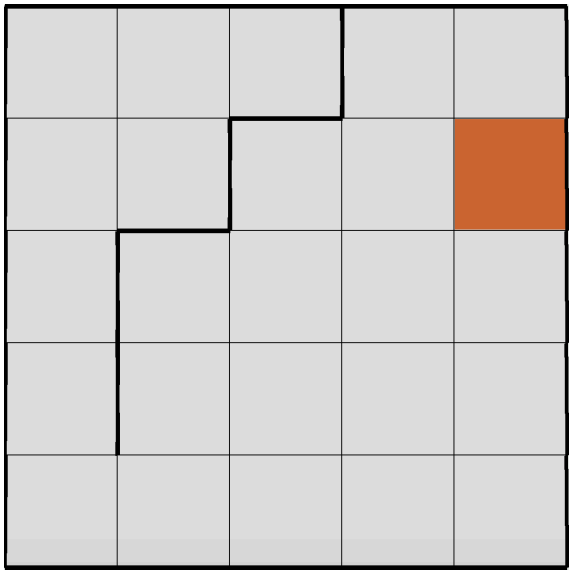
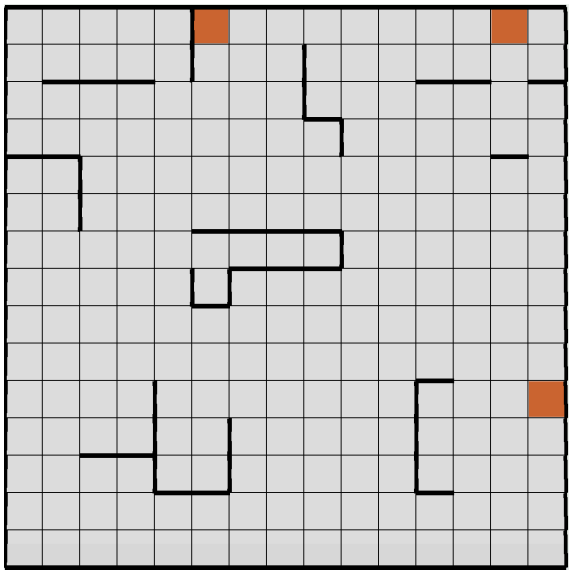
Markov Decision Processes

Tiffany Xu

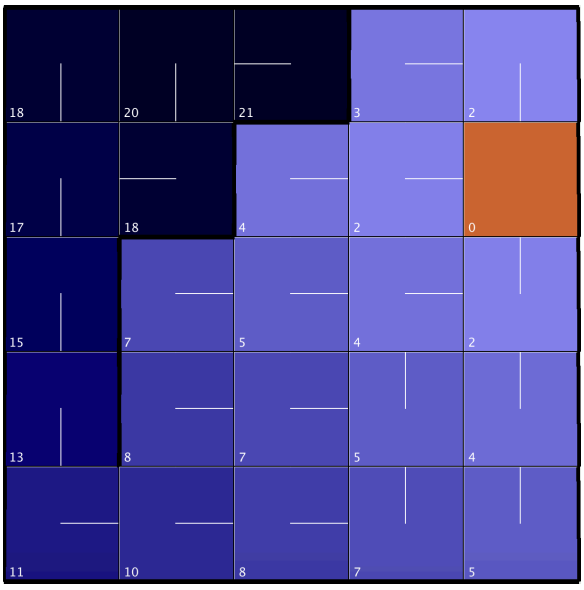
GridWorld MDP

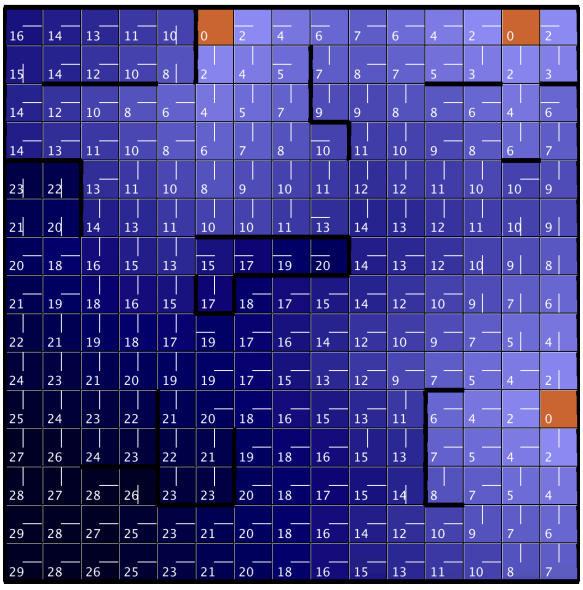




The two gridworlds that will be used to run planning and reinforcement learning algorithms are shown above. The first one is a simpler 5x5 sized world (25 states) that was inspired by situations in which I get lost and cannot get out until I backtrack to where I started. The second is a larger, 15x15 sized world (225 states) with many more walls, and multiple goal states. The penalty for being in each state, except for the goal state, is -1, thus incentivizing the agent to reach the goal state. Additionally, there is a -50 penalty for bumping into the walls so the agent will try to avoid hitting any walls. It will be interesting to see how value iteration, policy iteration, and Q-learning differ for these two different types of gridworlds, with differing sizes, walls, and number of goal states.

Value Iteration





Value iteration is a planning algorithm that is rooted in the idea of values or utilities. The utility of a state V(s) is the expected long-term discounted reward given that the agent starts at state s and follows some sequence of actions to the goal state. If our goal is to maximize our rewards, it makes sense for the agent to move to the neighboring state with the highest utility. These movements from start state to higher utility states until reaching a terminating goal state form the basis for policy making in value iteration [1].

The final policies that value iteration converged to are shown above. Below is a table showing the number of steps and time in milliseconds needed to converge to the policy depending on the PJOG. PJOG alters the transition model; a PJOG of 0.2, for example, means that 80% of the time the agent moves in the intended action and 20% of the time the agent goes into one of the other three neighboring states [1].

For both worlds, the number of steps needed to converge increased as the PJOG increased up until 0.8, after which the steps started to decrease. The initial increase in steps to convergence make sense, since going in unintended suboptimal directions at a higher rate will result in taking more steps to convergence. A probable reason for why the steps decrease with a PJOG from 0.8 to 1 is that because in both worlds there are situations in which the agent could get stuck behind a wall that separates the agent from a goal state, it would actually be more optimal to take a seemingly suboptimal action (i.e. bump into a wall now and get a -50 penalty) than to further explore these walled off areas that seem to be better because of the smaller -1 living reward, but that will eventually end up with bumping into a wall and acquiring another -50 anyways.

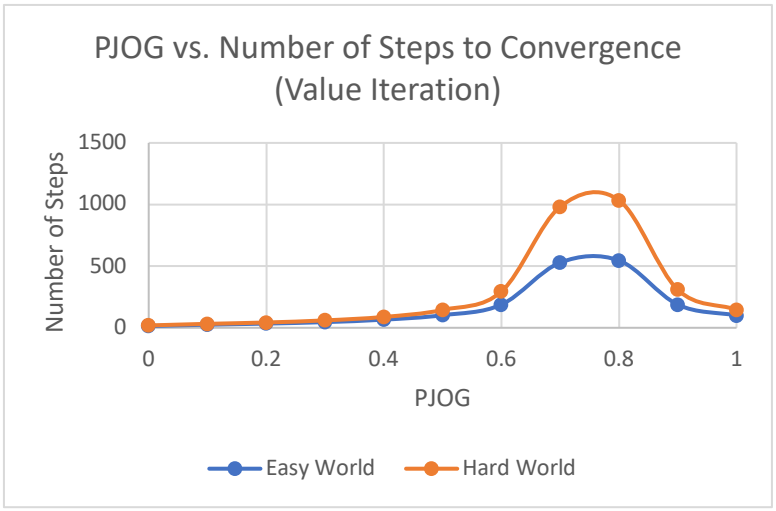
Table 1. PJOG vs. Steps and Time to Convergence for Value Iteration (Small World)

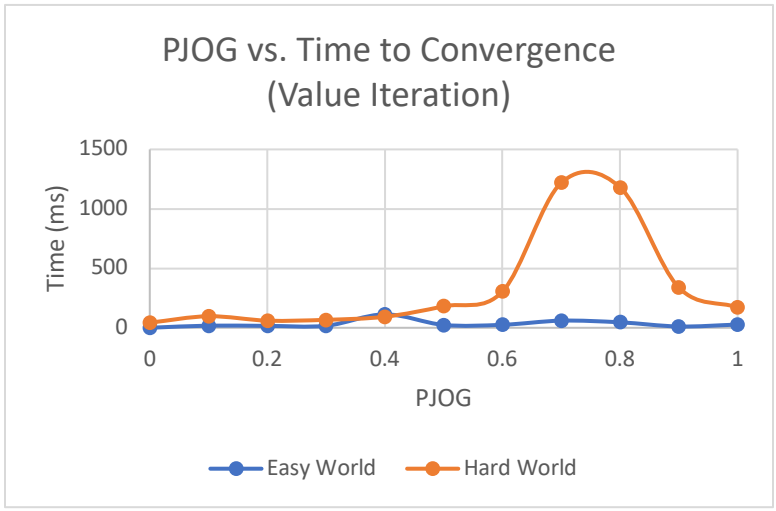
PJOG	Steps	Time (ms)
0	14	0
0.1	25	18
0.2	34	16
0.3	46	17
0.4	66	113
0.5	102	24
0.6	188	25
0.7	529	60
0.8	546	46
0.9	188	11
1	100	27

Table 2. PJOG vs. Steps and Time to Convergence for Value Iteration (Hard World)

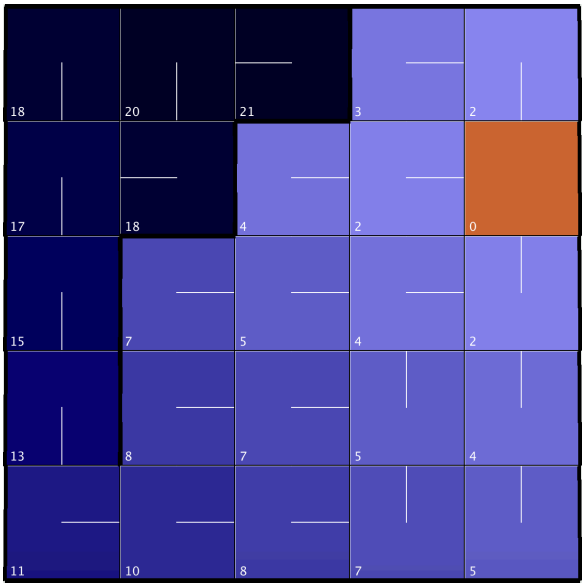
PJOG	Steps	Time (ms)
0	19	43
0.1	32	97
0.2	43	60
0.3	60	67
0.4	88	93
0.5	145	181
0.6	294	305
0.7	982	1219
0.8	1035	1181
0.9	309	339
1	147	172

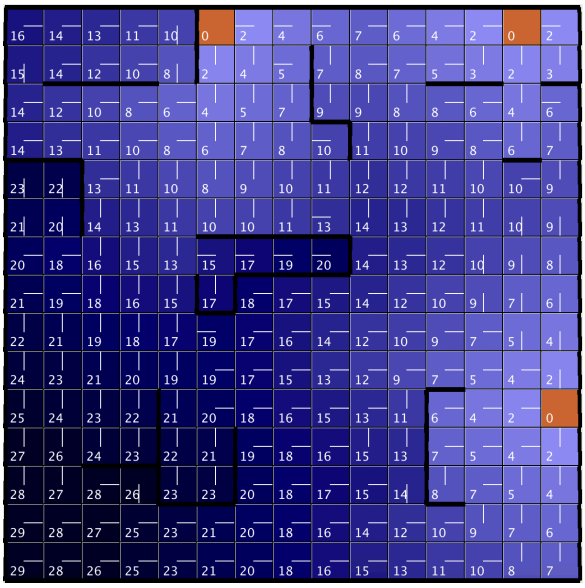
Below we see the comparison between steps and time to convergence for the easy and hard world. For the steps to convergence, the pattern is the same for both worlds, but the hard world is exaggerated since its greater size and complexity will expectedly take more steps to convergence. The time to convergence, however, seems to remain relatively constant for the easy world, but increases and then decreases in a similar pattern as the PJOG vs. steps for the hard world. For larger worlds, the more steps taken, the larger the impact it has on time; the number of steps seem to directly correspond to the time taken. On the other hand, small worlds only go up to a relatively small number of steps, up to which the time to converge do not differ much.





Policy Iteration





Policy iteration is another planning algorithm, but one that starts with a random policy. The utilities of each state are computed and updated based on this policy, and the optimal action in each state is then updated based on the utility. These changes in actions then result in a different policy, from which new utilities are computed and updated—this loop continues until the policy no longer changes [2]. The final policy that policy iteration converged to is shown above and is exactly the same policy found by value iteration for both worlds.

Unlike the patterns found in value iteration, there is no general trend for either worlds depending on PJOG, but the steps required to convergence remain in a much smaller range. Whereas the steps for the small world to converge range from 14 to 546 in value iteration, they only range from 2 to 8 in policy iteration. This range is reduced even more for the hard world, from 19 to 1035 in value iteration to 3 to 13 in policy iteration.

The reason policy iteration takes much less steps to converge to a policy is because policy iteration stops once the policy no longer changes. Value iteration, on the other hand, finds the policy indirectly, based on the utility of each state; it determines the final policy only after all the state utilities have converged up to a certain precision. Policies stop changing many steps before the utilities converge, thus policy iteration takes less steps to find the policy.

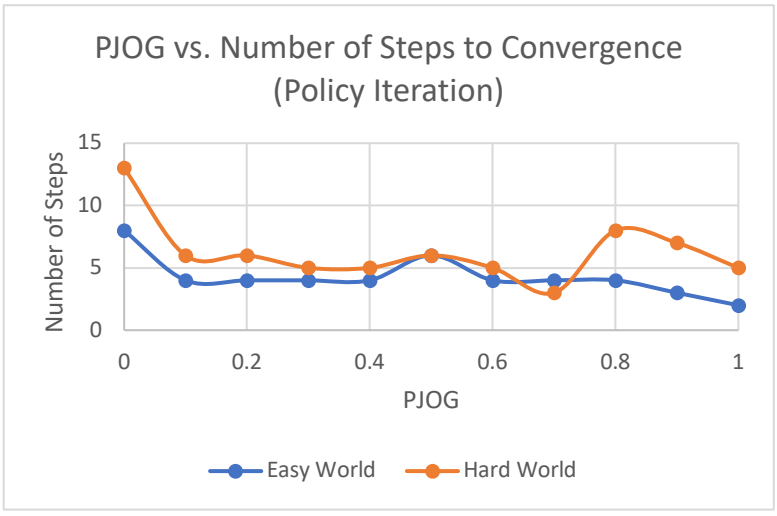
Table 3. PJOG vs. Steps and Time to Convergence for Policy Iteration (Small World)

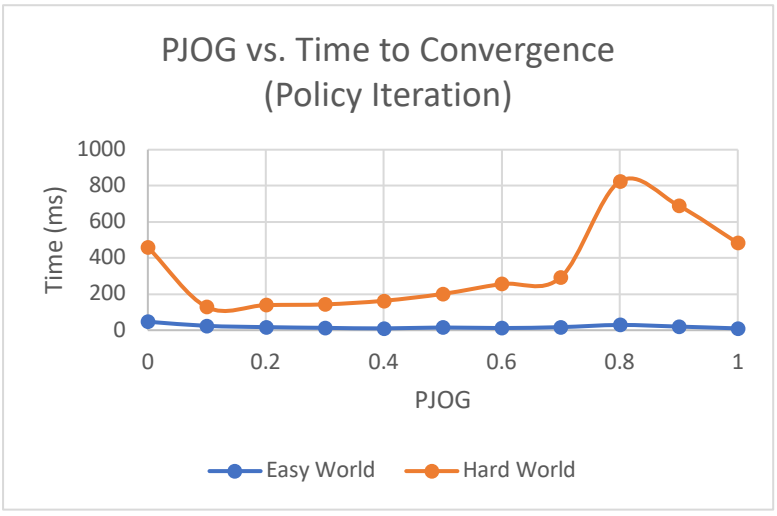
PJOG	Steps	Time (ms)
0	8	47
0.1	4	24
0.2	4	17
0.3	4	13
0.4	4	10
0.5	6	15
0.6	4	12
0.7	4	17
0.8	4	29
0.9	3	20
1	2	10

Table 4. PJOG vs. Steps and Time to Convergence for Policy Iteration (Hard World)

PJOG	Steps	Time (ms)
0	13	460
0.1	6	128
0.2	6	139
0.3	5	143
0.4	5	163
0.5	6	201
0.6	5	256
0.7	3	293
0.8	8	824
0.9	7	689
1	5	483

When comparing the steps to convergence for the easy and hard world, we see that overall, the easy world takes less steps, but the difference between the worlds at the biggest gap is much smaller than the difference in value iteration. Similar to value iteration though, the time taken for convergence remains the same for the small world no matter the PJOG but increases for the hard world up to a PJOG of 0.8. This increase in time at a PJOG of 0.8 followed by a decrease is directly related to the number of steps taken to converge. Once again, with a greater magnitude of steps, the steps directly affect the time taken to converge.





Q-Learning

In most cases, we will not have all the information necessary to plan our actions in each state. In the value iteration and policy iteration planning algorithms, we knew the states, actions, rewards, and transition model; these algorithms could find an optimal policy without the agent ever actually moving in the gridworld.

Q-Learning is a reinforcement learning algorithm that also finds a policy, but in the case where we don't know the rewards or transition model. Thus, the agent must learn as it goes, taking actions in each state to learn what the corresponding rewards and transition models are. This also brings up an exploration versus exploitation problem, where the agent must decide whether it should take the best action given the current learned information, or explore and collect more data, which could lead to a greatly negative reward, or a better optimal action.

To analyze the exploration-exploitation problem, I will modify the ε-greedy value and examine its impact on the resulting policy, given different number of iterations. ε-greedy is a method used to encode how much the agent should explore or exploit. Given an ε value, for example 0.3, the agent will perform a random action with 30% probability and perform the exploited action (i.e. current best) with 70% probability. A PJOG of 0.3 and a learning rate of 0.7 is set constant throughout this analysis. In this context, an iteration is an episode from start state to goal state.

The results are show in the tables below based on epsilon value and number of iterations. The number in the parenthesis next to the iterations is the number of incorrect actions in the policy compared to the policy found by the planning algorithms. As expected, as the number of iterations increases, the fewer mistakes the policy makes and the closer the Q-learning policy gets to the actual optimal policy.

Unfortunately for the small world, with each ten-fold increase in iterations, the policy only barely improves. I ran Q-learning to up to 10,000,000 and the policy still had several mistakes.
When looking at the differences between the epsilons within the same number of iterations, we see that an epsilon of 0.1 and 0.3 both perform around the same but perform better than an epsilon of 0.5. This means that in this world, an agent that exploits more than it explores can find an optimal policy faster than one that explores more than it exploits.

For the hard world, there is more improvement with each ten-fold increase in iterations, but that is also because there is a lot more room for improvement. Interestingly, an epsilon value of 0.5 is actually optimal, consistently producing less errors in the policy than epsilons of 0.3 or 0.1. This pattern is opposite from the case in the small world, where the smaller epsilon values did better. Since the hard world is so much bigger, it benefits a lot more from more exploration. Thus, an agent in a larger and more complex world finds an optimal policy faster when it explores just as much as it exploits.

I did not have a precise way to measure runtimes for the Q-learning algorithm, but there was an obvious increase in runtime with more iterations; this was especially apparent in the latter ten-fold increases in iterations. There was no obvious difference in runtimes with differing epsilon values for the same number of iterations though. These trends in runtime are the same for both the small and hard world, and the hard world, with its greater size and complexity, expectedly look longer to run than the small world.

Table 5. Iterations and Epsilon vs. Resulting Policy for Q-Learning (Small World)

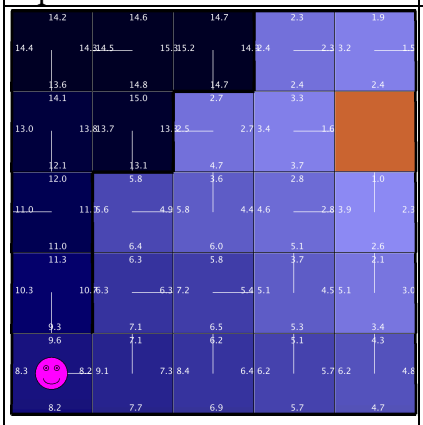
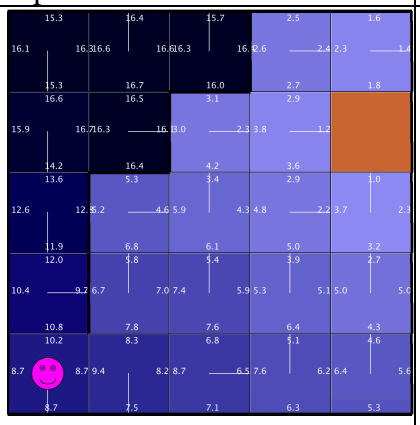
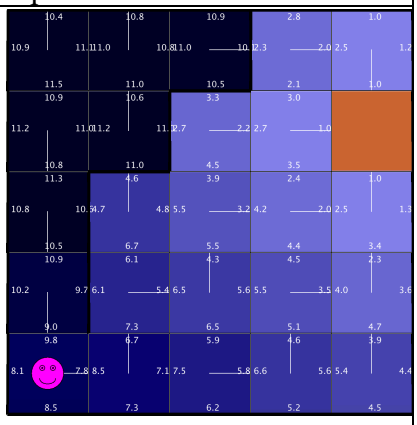
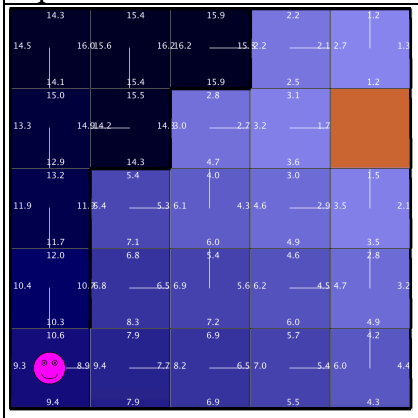
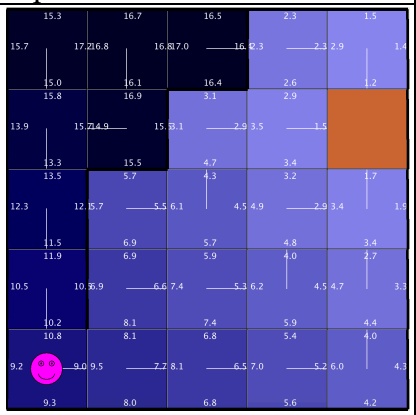
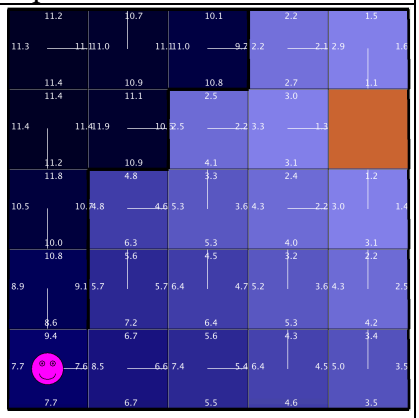
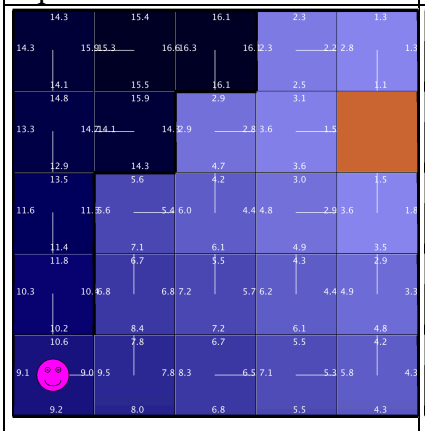
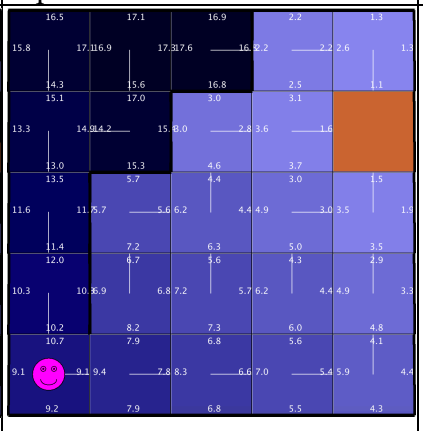
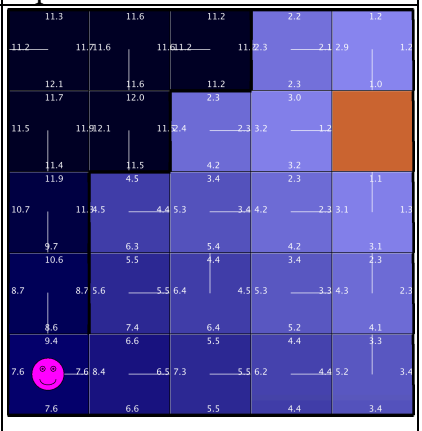
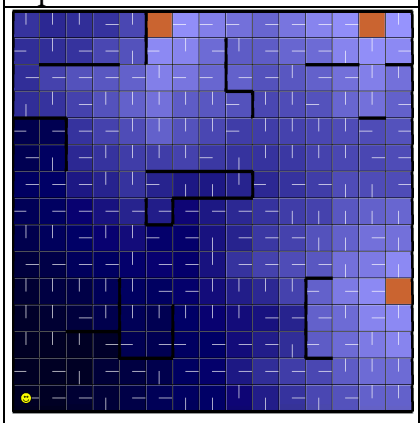
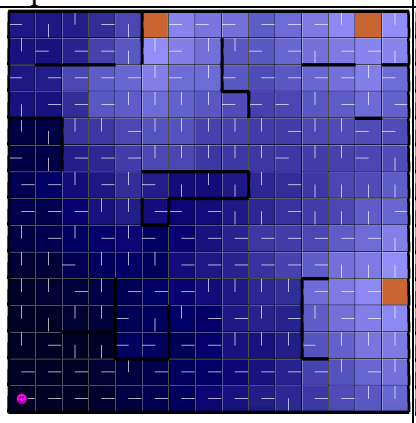
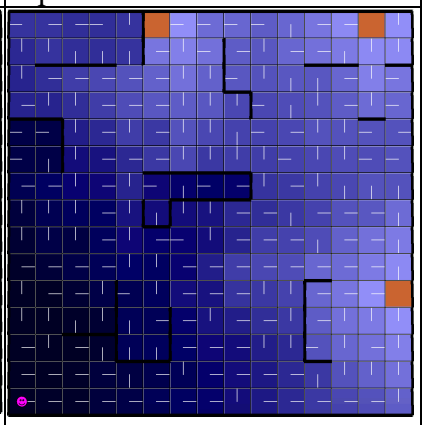
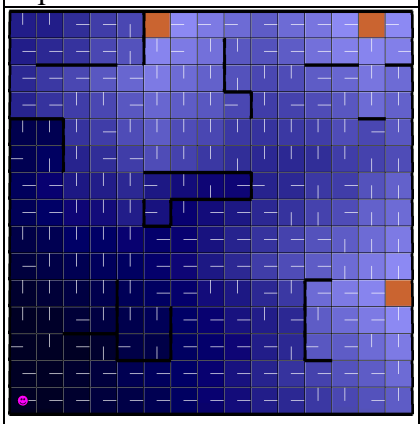
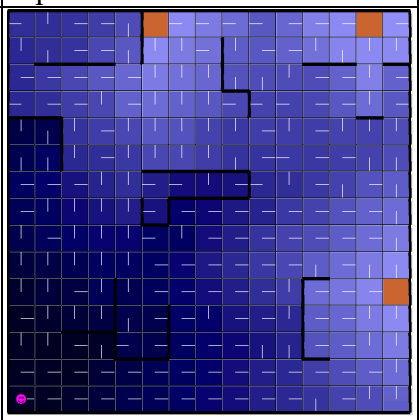
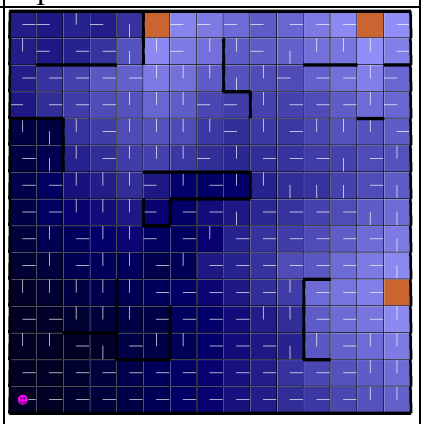
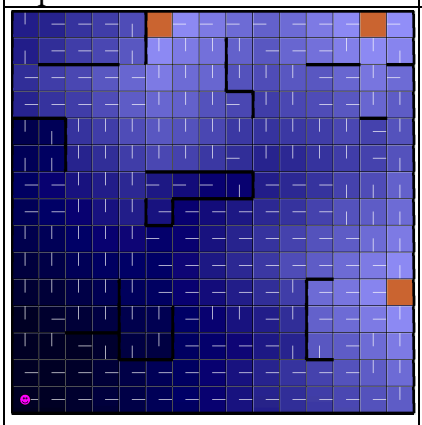
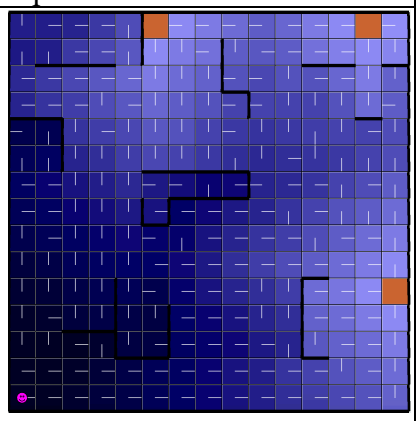
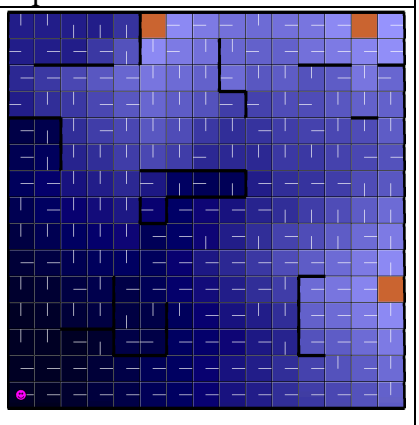
1,000 Iterations (9) Epsilon = 0.5	1,000 Iterations (10) Epsilon = 0.3	1,000 Iterations (8) Epsilon = 0.1
		
10,000 Iterations (5) Epsilon = 0.5	10,000 Iterations (3) Epsilon = 0.3	10,000 Iterations (7) Epsilon = 0.1
		
100,000 Iterations (7) Epsilon = 0.5	100,000 Iterations (6) Epsilon = 0.3	100,000 Iterations (5) Epsilon = 0.1
		

Table 6. Iterations and Epsilon vs. Resulting Policy for Q-Learning (Hard World)

1,000 Iterations (81) Epsilon = 0.5	1,000 Iterations (82) Epsilon = 0.3	1,000 Iterations (95) Epsilon = 0.1
		
10,000 Iterations (50) Epsilon = 0.5	10,000 Iterations (55) Epsilon = 0.3	10,000 Iterations (60) Epsilon = 0.1
		
100,000 Iterations (17) Epsilon = 0.5	100,000 Iterations (36) Epsilon = 0.3	100,000 Iterations (57) Epsilon = 0.1
		

Further Work

To continue to analyze this MDPs and reinforcement learning, I would try to use another package that would allow for different exploration methods, such as Boltzmann or random exploration. It would be interesting to see if these other methods would do better than ε-greedy and compare those results with the ones found here. I would also like to run the Q-learning algorithm for both worlds until it arrived at or really near the optimal policy. I was unable to do that with the code I used, because with higher iterations, especially with the hard world, the application seemed to freeze. Knowing how many iterations Q-learning would take to reach the optimal policy can show just how much harder Q-learning is compared to value and policy iteration; this is important because in many and probably most real life situations, we will not know the rewards or transition models, and will need to use reinforcement learning algorithms like Q-learning to learn to react in those environments.

References

[1] Mehta, V. and Kelkar, R. *Simulation and Animation of Reinforcement Learning Algorithm.* Carnegie Mellon University. Accessed 12 April 2019.

[2] Veloso, M. *Reinforcement Learning: Value and Policy Iteration.* Carnegie Mellon University Computer Science Department, http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15381-f01/www/handouts/111301.pdf. Accessed 12 April 2019.