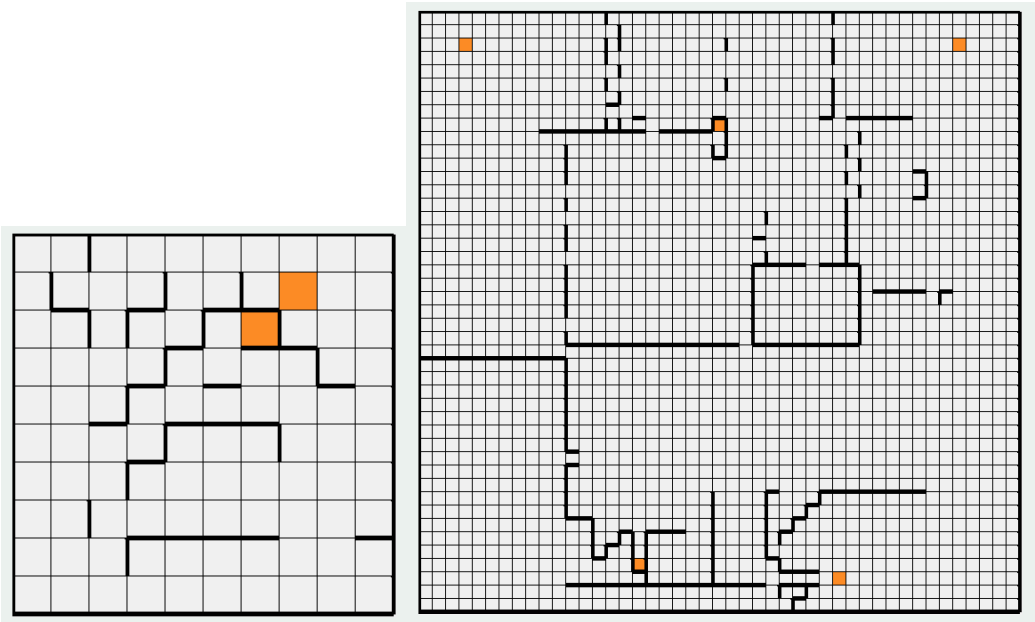
James Liu
Spring 2014, CS 4641
April 20th 2014
Markov Decision Processes and Reinforcement Learning Assignment

MDPs Chosen



For the purpose of this project, I created two gridworld mazes for a MDP based agent to navigate. For the agent, staying in any of the states gives a -1 reward, but colliding into a wall will penalize it heavily with a -50 reward. Reaching the goals (marked in orange), will terminate the game. With this construction, the agent will attempt to get to the goals as fast as possible, while avoiding walls. The transition function for these problems allows for all actions except the opposite of the chosen action. The probabilities in which each action is chosen is dependent on the PJOG parameter provided, which is essentially the probability that the agent will not do the originally intended action. If the agent attempts to move into a wall, it will stay in the current state it's in and take a -50 penalty.

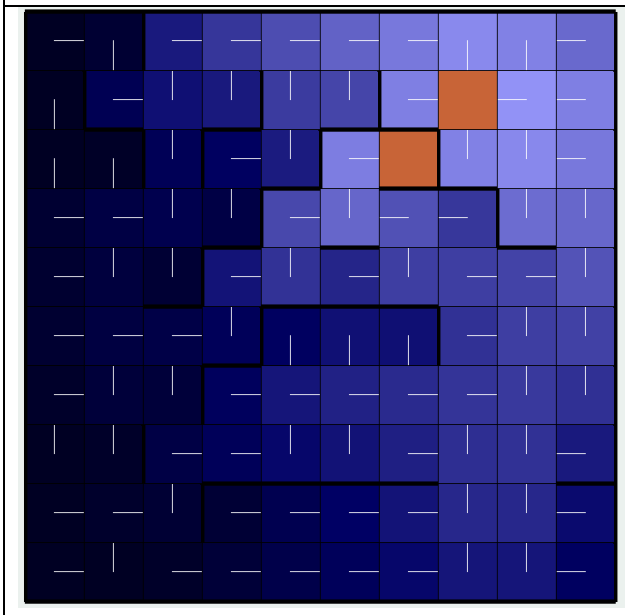
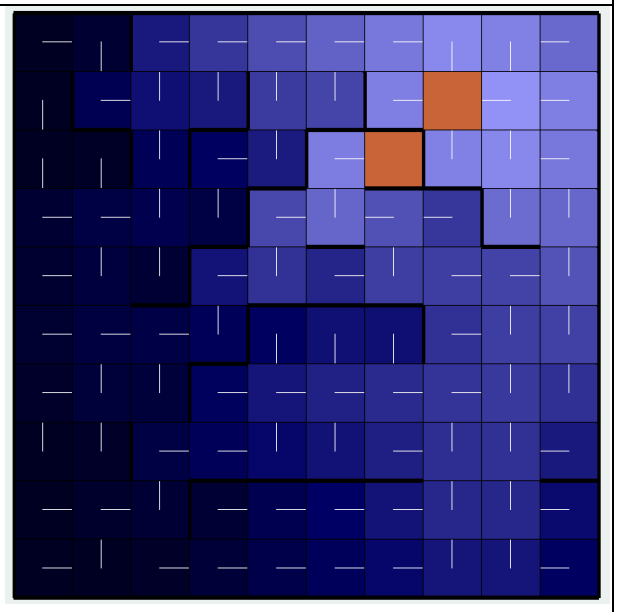
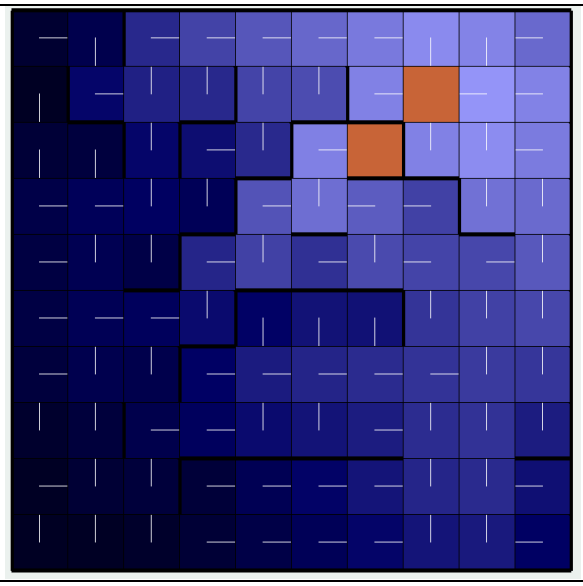
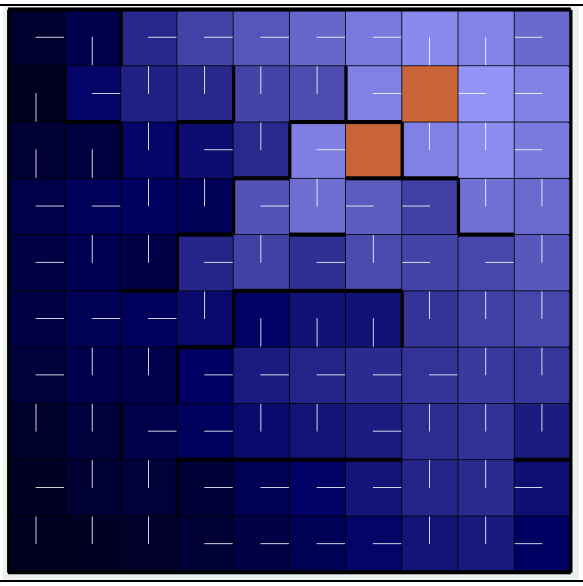
The first maze is 13x13, for a total of 169 states, two of which are goals that are of close proximity to each other, but is separated by a large number of walls. It is interesting because of the presence of multiple goals. It will be interesting whether the MDP will prefer to take the long, but wall avoiding path around the bottom to go for the top goal, or try to go for the higher risk goal surrounded by walls.

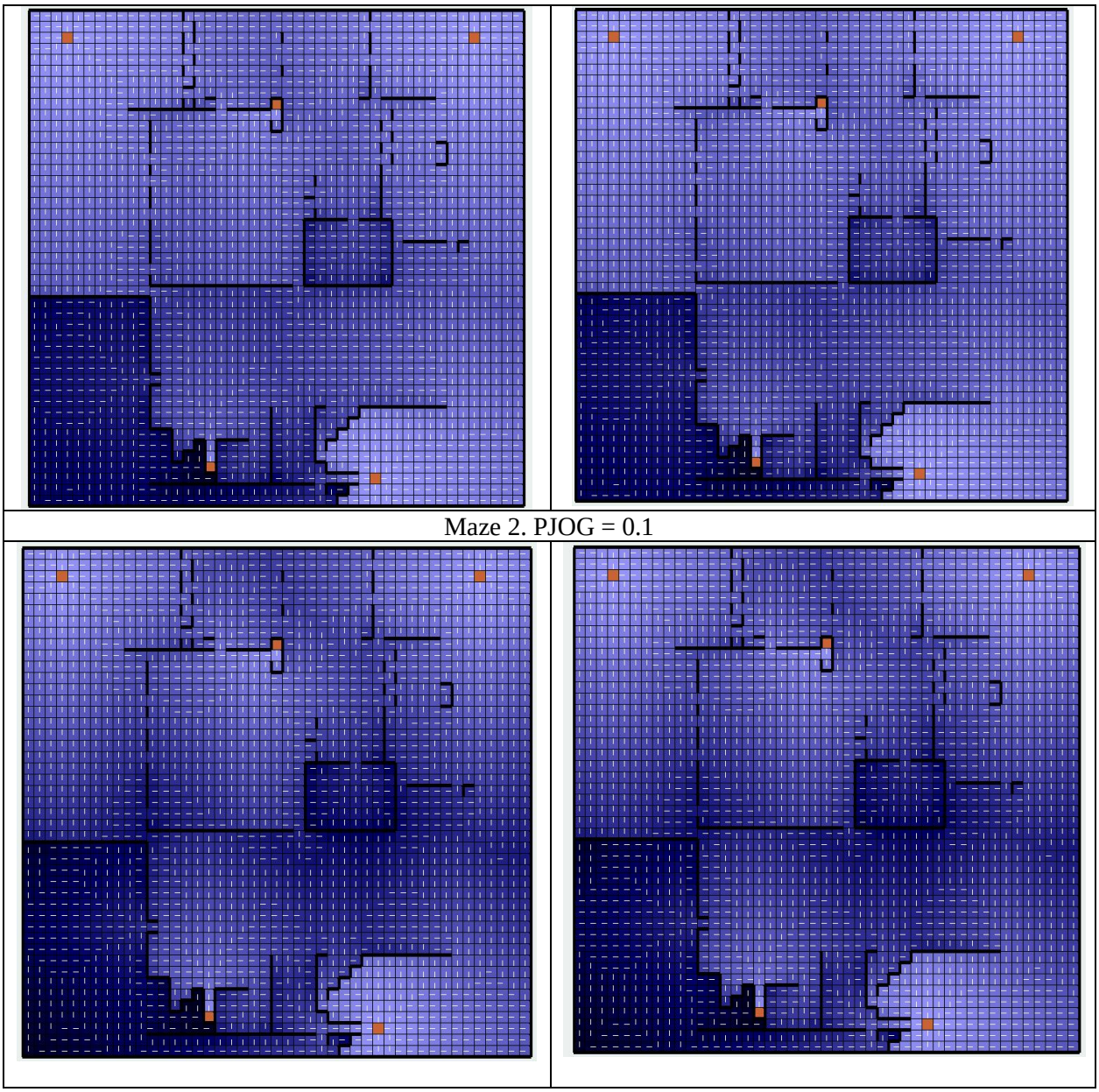
The second maze is 45x45, for a total of 2,025 states, with a total of 5 goals that are separated by a large distance. This is more interesting in terms of reinforcement learning algorithms, as the larger state space will show some problems of scalability with RL problems, and the walls between each goal will be interesting in that it might show the problem of exploration vs exploitation in RL problems.

Value Iteration vs Policy Iteration

Maze	Algorithm	PJOG	Total Time	Iterations Needed
1	Value Iteration	0.3	65ms	84
1	Policy Iteration	0.3	122ms	7
1	Value Iteration	0.1	39ms	42
1	Policy Iteration	0.1	162ms	6
2	Value Iteration	0.3	13063ms	154
2	Policy Iteration	0.3	48438ms	16
2	Value Iteration	0.1	5595ms	66

2	Policy Iteration	0.1	42204ms	22

Value Iteration	Policy Iteration
Maze 1. PJOG = 0.3	
	
Maze 1. PJOG = 0.1	
	
Maze 2. PJOG = 0.3	



Maze 2. PJOG = 0.1

First, a look at the numbers. It is very evident that policy iteration takes fewer iterations to converge under every case tested. This is an obvious result of how the two algorithms terminate. Value iteration will continue until no values of U change (within a given tolerance) after a given iteration, whereas policy iteration continues until the policy no longer changes. This would cause value iteration to continue, even though the actual actions taken by the agent a given state no longer changes, and thus require more iterations to converge. However, even though policy iteration requires fewer iterations, it took significantly longer timewise. This can be largely attributed to how value iteration only requires computation of the values of neighbor states ($O(n)$ in number of states per iteration), whereas policy iteration requires evaluation of the entire Markov chain from any start state ($O(n^3)$ in the number of states, per iteration).

In terms of differences between outputs between the two algorithms, there are no significant differences between both of them when it comes to the final resultant policy. However, even on the small maze, the final utility values for each state showed a few minor differences. This highlights, once again, the difference in termination condition for both algorithms: since policy iteration stops after the policy

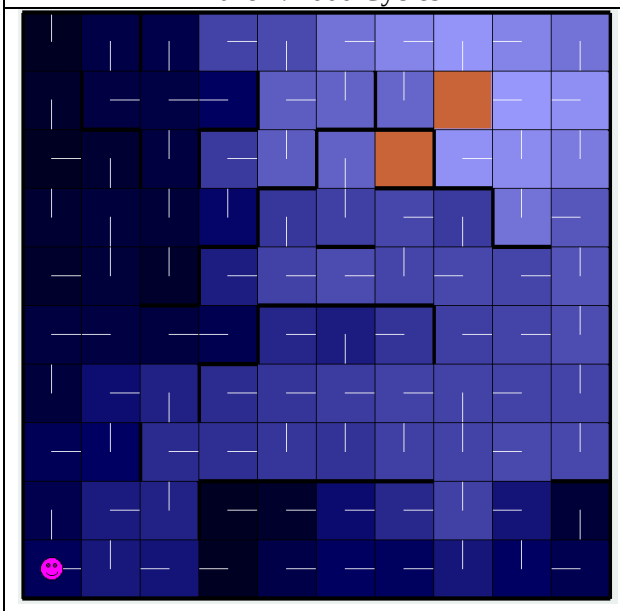
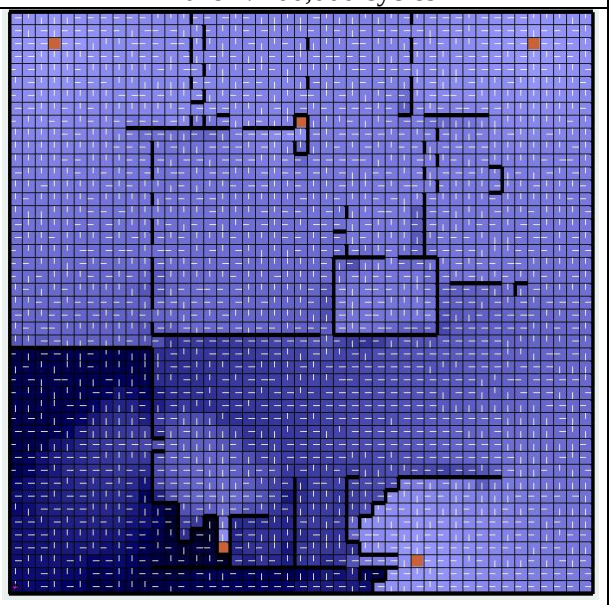
ceases to change, the utility values it computes for each state are not guaranteed to be correct, unlike value iteration, but will nonetheless converge to the right action to take.

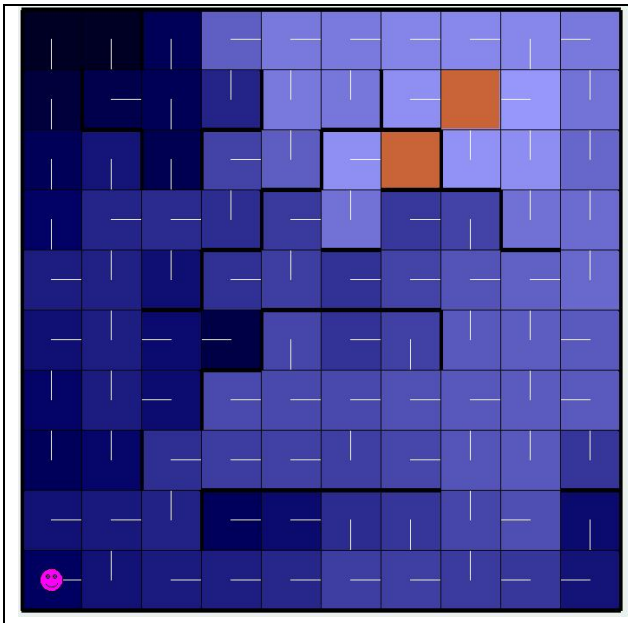
The number of states did significantly increase the time and iterations needed for both algorithms to converge, particularly with policy iteration. The large maze has about 12 times the number of states that the small maze does, but policy iteration took almost 300 times longer on it, timewise. More states mean more iterations needed to propagate the correct values to every state. For policy iteration this means it takes longer to compute the value of the Markov chain from each state, and thus each iteration takes longer. For value iteration, this means that it requires more iterations for the algorithm to converge to the exact utility values for each state.

Finally, I tested how changing the transition function of the MDP would affect the final policy, by modifying the PJOG value. As PJOG approached 0, the final policy increasingly became less wary of walls, eventually becoming willing to move directly parallel to it instead of moving into open space to avoid the chance of colliding into them. Additionally, as the agent became increasingly sure of its actions, the total necessary runtime/iterations decreased slightly, as the policy/values were less likely to change due to less noise in the transition function.

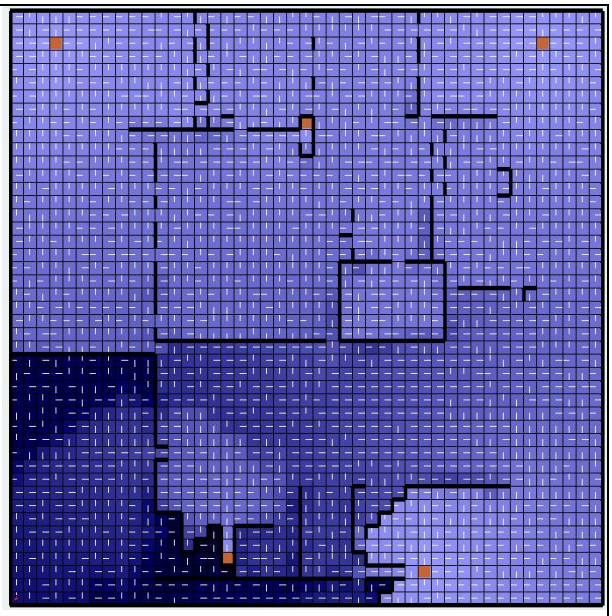
Q-Learning

These experiments were done with a PJOG of 0.3, decaying learning rate of 0.7, and an epsilon of 0.1.

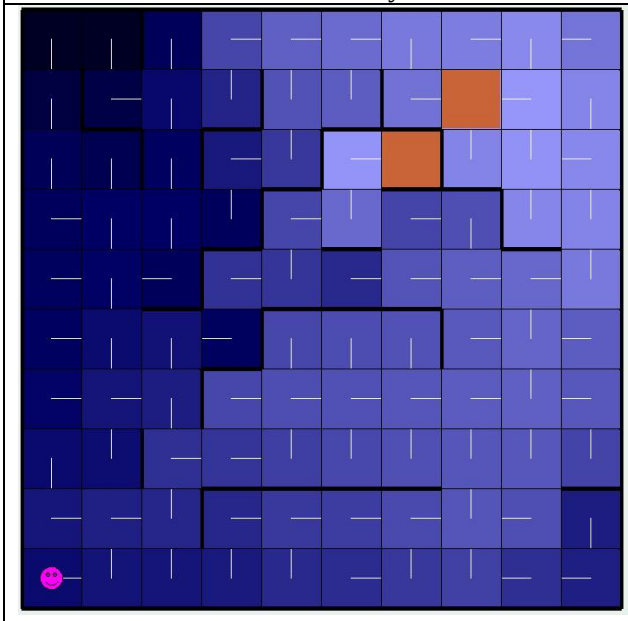
Maze 1. 1000 Cycles	Maze 2. 100,000 cycles
	
Maze 1. 5000 Cycles	Maze 2. 500,000 cycles

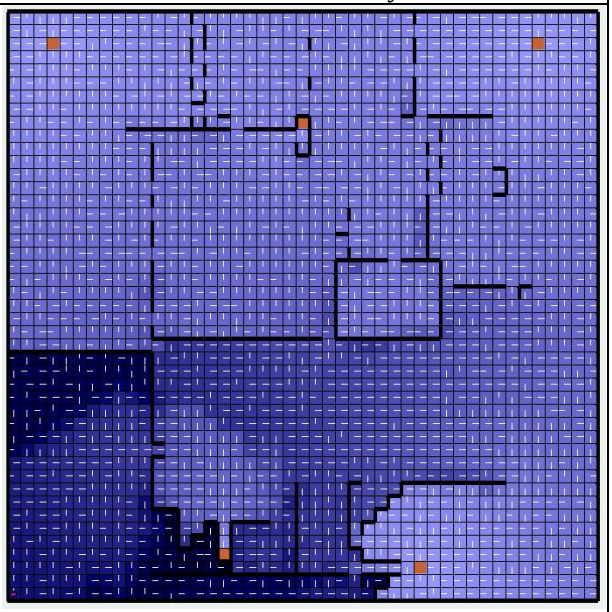


Maze 1. 10000 Cycles



Maze 2. 1,000,000 cycles

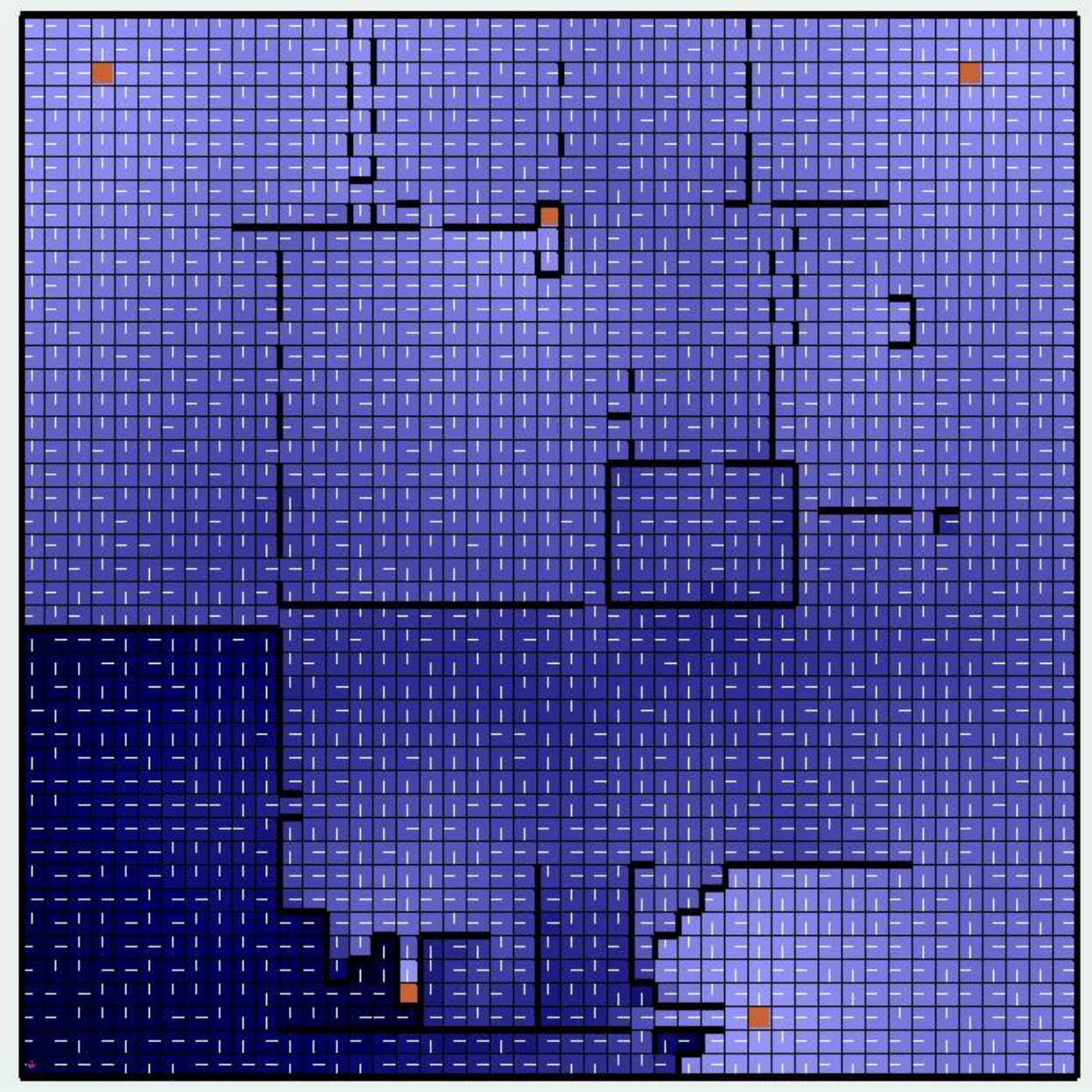




Compared to the value and policy iteration, Q-Learning seems to produce subpar results from the given samples. It took Q-learning 10,000 cycles of running the agent through the small maze all the way to the end to get something resembling the final result of value iteration and policy iteration, which took a full 10 seconds, whereas value iteration only required 63ms to converge. However, unlike value iteration, Q-learning did not need prior knowledge of the state space, or the transition model, yet still built an approximately optimal policy in reasonable time.

The large maze, however, took significantly longer to converge due to the increased state space size. It also shows one of the major dilemnas of RL: exploration vs exploitation. From the above samples, it can be seen that the parts with heavy gradients in value are areas where the agent has explored. Particularly the goal in the bottom left. The problem is set up such that the agent wants to end the game as fast as possible, and it seems that, with an epsilon of 0.1, the algorithm tends to greedily head towards the closest

goal. However, in terms of discovering what to do from the rest of the state space, the agent has rarely ventured toward the upper three goals. By turning up epsilon, the agent is tends explore than to exploit.



Above is a trial on the large maze only after 1,000 cycles, but with an epsilon of 0.99. The final result largely resembles the results from value/policy iteration, but it also takes signifigantly longer than when episolon was set to 0.1: 100,000 cycles at epsilon of 0.1 took approximately 10-11 seconds, which was equally as long as the 1,000 cycles at epsilon of 0.99. This increase in time is because of the agent's increased tendency to explore the state space, rather than exploit known goals. It is far more likely to not follow the saved policy, and thus wanders much more than if it were exploiting more

Conclusion

We see here three algorithms with dealing with MDPs: value iteration, policy iteration, and Q-learning. Value iteration is generally very quick at computing the optimal policy, but requires a large number of iterations to converge and requires full knowledge of the state space and model. Likewise, policy iteration is slower due to the need to evaluate the entire Markov chain from a given state, and still needs

full knowledge of the model and state space. Both tend to do better on more deterministic models that have low noise. Q-learning requires more time to converge, and suffers from the exploration vs exploitation dilemma, but requires no prior knowledge of the state space or the model. Exploration may lead to the agent taking less optimal actions, but as a result may find more optimal solutions later on. Exploitation causes the agent to utilize actions that are known to produce good results, but may cause it to converge to only local optima.