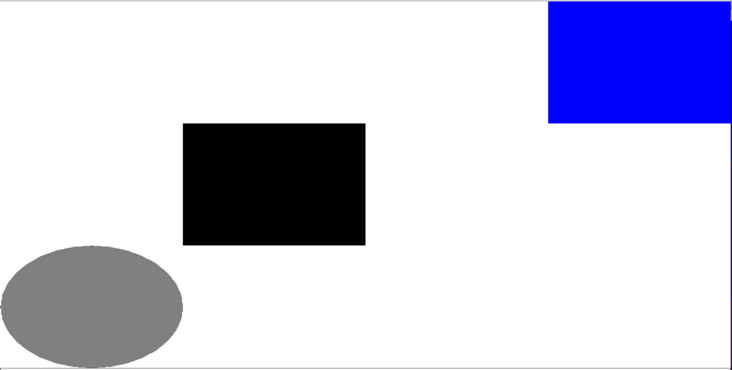
**CS 7641 Project 4: Markov Decision Processes**

Yaling Wu (ywu342)

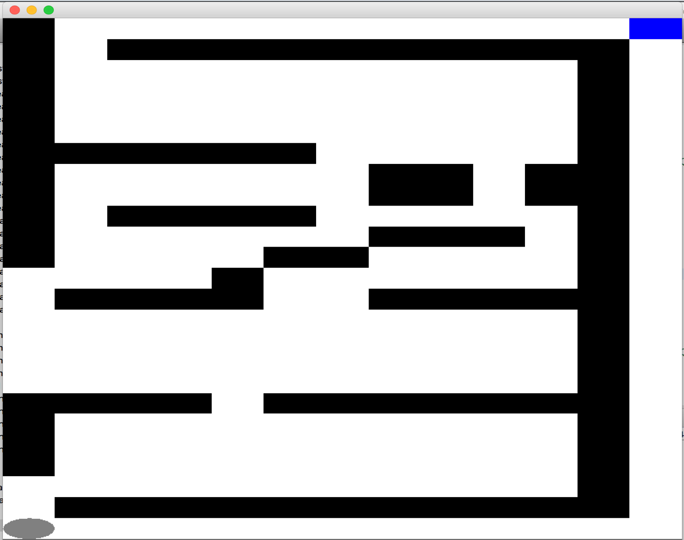
# MDP Problems

The two MDPs I selected to use are both GridWorld problems. The main goal is to find the best policy per state such that total reward is maximized. One important characteristic of this kind of problem that makes it interesting is its setting is intuitive in the context (has all the components an MDP possesses) and easy to understand. It would be easy to visualize the changes in policies and utilities of each state on the GridWorld. GridWorlds have a discrete state space and discrete action space. Convergence proofs and analysis will be easier to understand and visualize for discrete spaces. Each position is a state and properties of the states are stationary over time. Besides, two other important components of the problem are actions and rewards. Entering each state grants a sort of reward to the agent. At each state, an agent can move toward all four directions but choosing an action will render a stochastic movement. Thus, given each state and action, there is a probability for each new state cell adjacent to old state. With this kind of setting, GridWorld problems can be linked to some real life problems like cleaning robots or self-driving vehicles that try to get to a destination while avoiding obstacles on the way.

One example is shown below (my simple grid problem). Grey circle refers to the initial position of our agent. The black block refers to a piece of wall. Walls are not really a state. Heading to a wall will cause the agent to stay in the same state in the next time step. The blue square indicates a terminal state where entering the state gives a reward of 10. The orange square indicates another terminal state where a reward of -10 will be triggered. All the white space refers to normal states of the world. Each normal state has a reward of -1. The probability of the agent gets to move in the same direction as it has chosen is 80%. The other 20% of the time, the agent will move in a random direction. The whole state space is 3\*4-1 = 11. The whole action space is 4. An optimal path of the simple gridworld is shown in blue arrow. This makes perfect sense because going in the yellow dashed line would have a higher probability of falling into orange square. One other interesting thing about this layout of the gridworld is that if I change all normal state rewards to a very large negative value like -1000. Agent would prefer a route that starts at the right of initial position in that case to get done with the game as soon as possible.



Below is my second choice of the GridWorld. It has a similar layout as the last one but It has a much larger state space: 229. There are much more roadblocks and detouring in the way if agent chooses to enter the left entrance of the maze and on the right, there is only one way out. Action space, rewards and stochasticity all stay the same as last problem. I designed this layout in a comparison with the last one for the algorithms because the optimal path looks drastically different. In this graph, the optimal path would be to bypass the whole zigzagging maze of the wall-surrounded block and head straight into the bad terminal state (-10). This makes it more interesting to look at and do the analysis on. In addition, I would like to see how increasing the state space would impact all the algorithm performances.



For all the algorithms, discount factor stays the same: 0.99. Converging delta is always 0.001.

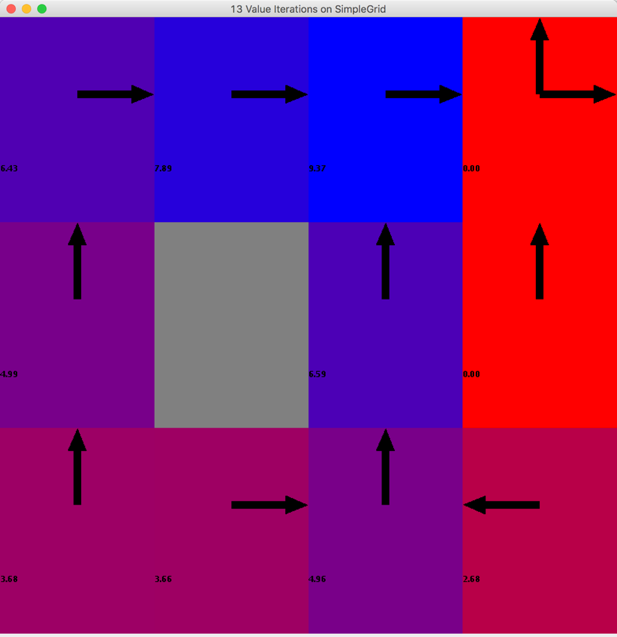
# Value Iterations

For this experiment, I ran value iterations on each MDP problem with number of iterations ranging from 1 to 100. And I set the converging threshold to be 0.001. So the algorithm would stop when utility changes between time steps fall below 0.001 even if the assigned iteration number has not reached.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MDP | Iterations to Converge | Time Used for Planning (ms) | Total Reward of an Episode | Number of Steps in an Episode |
| Small/Simple | 13 | 4.63565 | 4.0 | 7 |
| Large | 61 | 155.422292 | -61.0 | 52 |

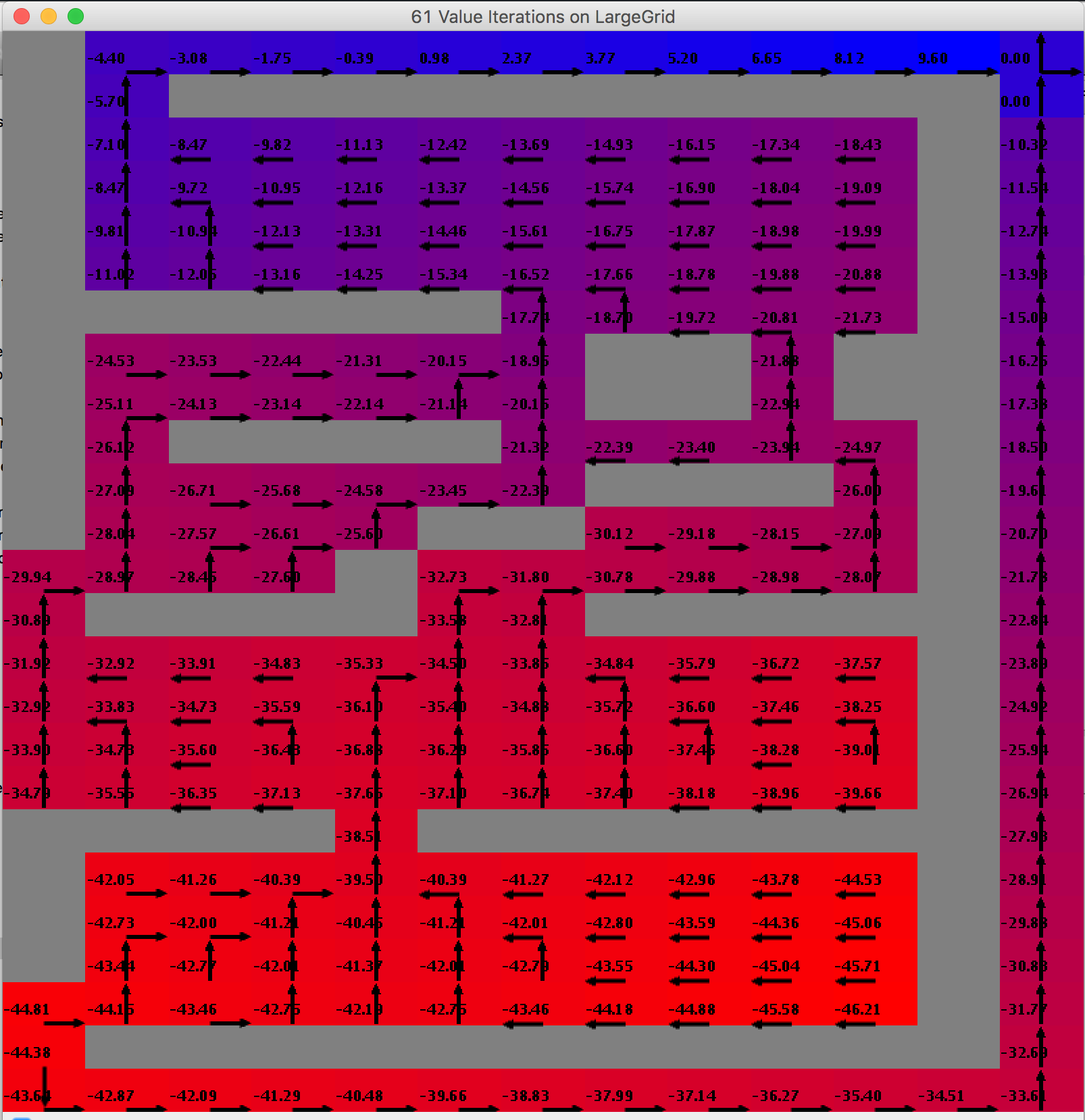
As shown in the table, small grid takes only 13 iterations to converge. It also takes only 4.63565 milliseconds to converge. On the contrary, large grid takes more iterations and more time in planning to convergence given the size of the problem. Large grid has more states to calculate and spreading of the true utilities from the terminal states would take longer time. Above table also shows how much reward is granted in an example episode given an optimal policy in each problem. When calculating the total reward of an episode, I did not discount future rewards along the way. From the results, we can see that while an agent can still get a positive total reward in small grid, it can only get a high negative reward in the large one. Because the +10 at the terminal is still able to offset the accumulated negative rewards from earlier steps. However, it takes many steps for the agent to reach any terminal state in any path. It collects too many negative rewards such that a +10 at the end cannot cancel all the negativity out.

## Small Grid



Above figure shows the optimal policy derived from converged utility values of all states (because value iteration is guaranteed to converge to the optimal solutions given there is no limit on max number of iterations). Just as indicated in the problem statement, the optimal path would be going up, up, right, right, and right to the +10 terminal state. It is reasonable that the bottom path to the right is suboptimal because there will be a chance to fall into the -10 terminal state.

## Large Grid



This shows the final utilities and corresponding policies of large gridworld. The optimal path is to go through the right route to the -10 terminal state. In this large grid, it would not be wise to go through a long route in the maze to the +10 terminal state because resulting accumulated negative rewards cannot be cancelled out by the final small positive reward. It would be less of a loss if quickly ending the game vis the right route.

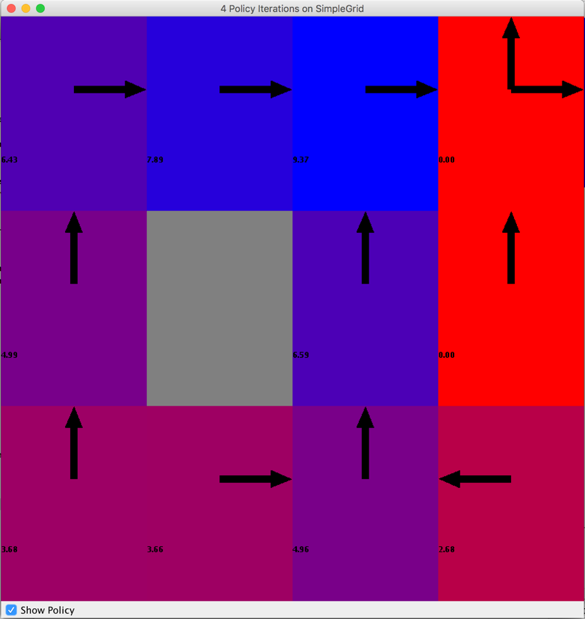
# Policy Iterations

For this experiment, I ran policy iterations on each MDP problem with the number of iterations ranging from 1 to 100. And I set the converging threshold to be 0.001. So the algorithm would stop when utility changes between time steps fall below 0.001 even if the assigned iteration number has not reached.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MDP | Iterations to Converge | Time Used for Planning (ms) | Total Reward of an Episode | Number of Steps in an Episode |
| Small/Simple | 4 | 48.598941 | 5.0 | 6 |
| Large | 13 | 1681.468846 | -58.0 | 49 |

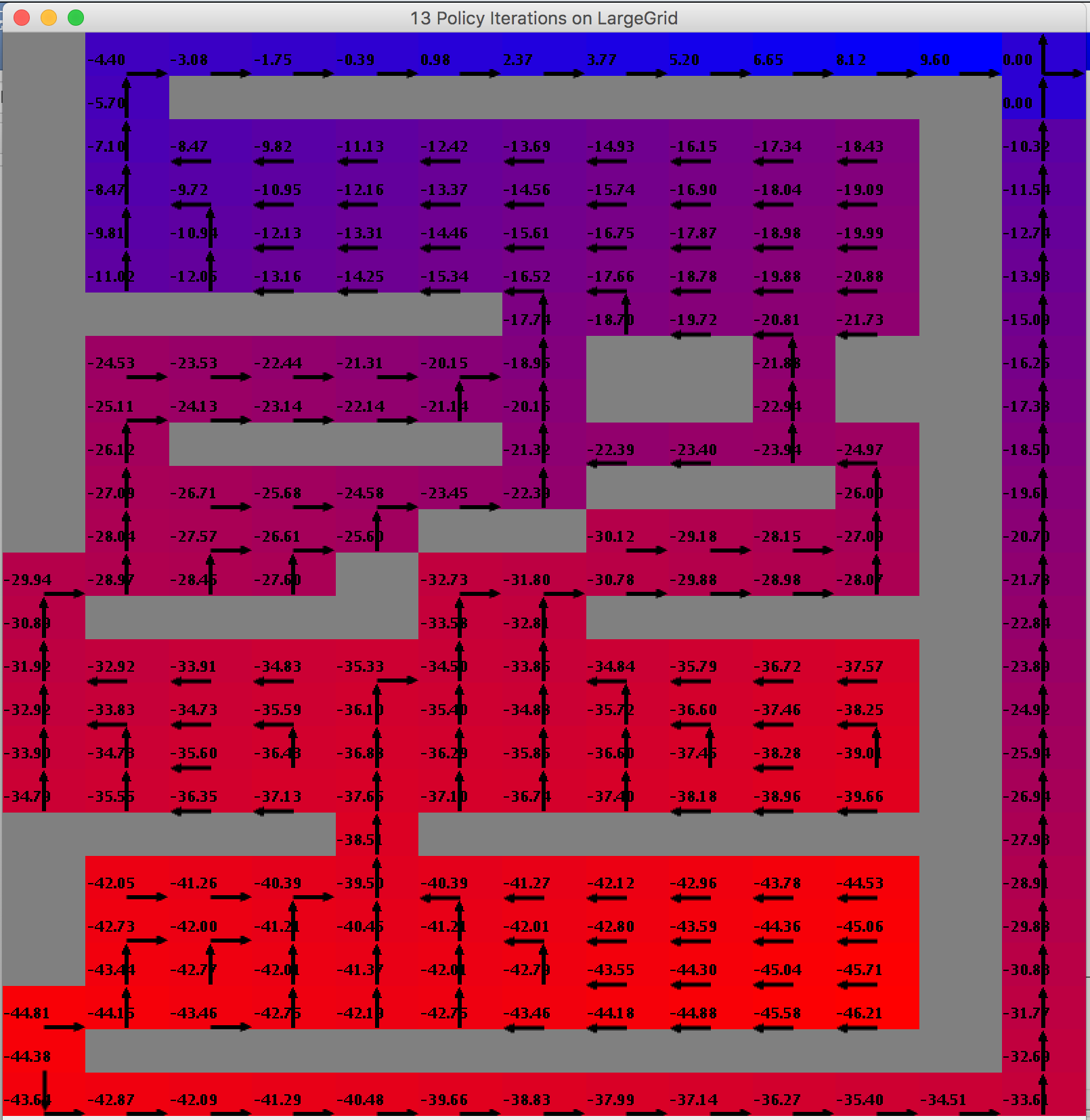
Similar to the results of those of value iterations, it takes more iterations and time for large gridworld to converge than small gridworld due to its larger size. Thus the bigger the state space is, the longer it takes to converge. Episode metrics here are different from those of value iteration because this is another instance of episode which can be different every time even under the same policy. It is also worthy to note that in general, policy iteration takes less iterations to converge than value iteration but the time it uses to converge is longer. This could be due to the fact that at each iteration of policy algorithm, much more work is needed like evaluating current policy by solving the N linear equations (N is the state space size) and updating policies. On the contrary, value iteration algorithm just updates utilities per iteration and computes policies at once in the end. In addition, iterating over the policy space usually makes a bigger move towards the optimal solutions.

## Small Grid



After policy iteration converges, it gives the same policy and same final utility values as the value iteration. Both policy iteration algorithm and value iteration algorithm are guaranteed to converge to the optimal solutions given unlimited number of iterations. Because their goal is the same: to use Bellman equation to calculate utilities of each state and thus update optimal policies from utilities. They are different in that they use different ways to solve the N equations of N unknowns. Value iteration solves this nonlinear problem with iterations but policy iteration solves it through iterations over updated policies and turns bellman equations into N linear equations at each iteration.

## Large Grid



Similarly, in this problem, policy iterations converge to the same utilities and same optimal policies as the corresponding value iterations. The reasons are depicted above in the small grid example.

# Effect of State Size on Planning Time

Here this graph shows how planning time grows quickly (exponentially?) as the size of state space increases. Used time of value iteration algorithm is not as affected by the size as much as the policy iteration. Again, at each iteration of policy algorithm, a lot more computations are required (both utilities and policies of all states are updated) than each iteration of value algorithm.

# QLearning

I chose QLearning as the reinforcement learning algorithm in this assignment. Again the convergence delta of Q-functions is set to 0.001. The exploration strategy used is the default one: EpsilonGreedy epsilon = 0.1 (little exploration). In the end of each problem section, different exploration strategies are experimented and analyzed.

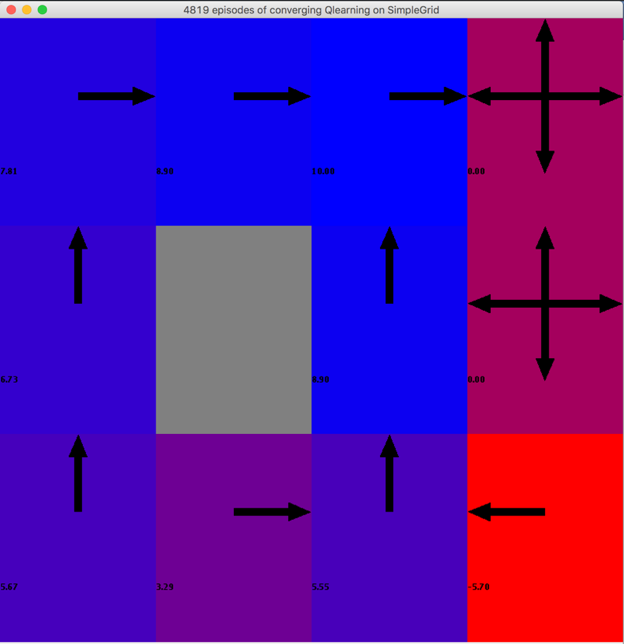
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MDP | Iterations to Converge | Time Used for Planning (ms) | Total Reward of an Episode | Number of Steps in an Episode |
| Small/Simple | 4819 | 152.176531 | 6.0 | 5 |
| Large | Integer.max | - | - | - |

As shown in the above table, QLearning requires significantly higher number of iterations to complete the convergence compared to previous two iterations. QLearning also takes much longer time to converge (would run out of heap space as well). Especially for the large gridworld, the state space is so big that convergence was not reached when I set the max delta to be 0.001.

## Simple Grid

As more number of episodes are simulated in QLearning planning, more information is learned about the underlying transition model. A better policy will result from the learning. This is why the above graph shows an overall increasing curve. Due to a fairly good policy, the total reward for each episode keep increasing until it reaches at around 30 episodes. So this shows in simple gridworld, it does not take long to converge to an optimal policy, even though best utility values have not been reached 30 episodes based on previous findings (at least 4800 episodes are needed to have the max utility change falls below 0.001).

Below figure shows a snapshot of the final policy when QLearning converges. The utilities are not the same as those from value iteration or policy iteration because QLearning can only approximate the underlying transition model to get a close estimated utility value, unlike the other two algorithms. However, the optimal policy it got is the same as previous two algorithms.



### Different Exploration Strategies

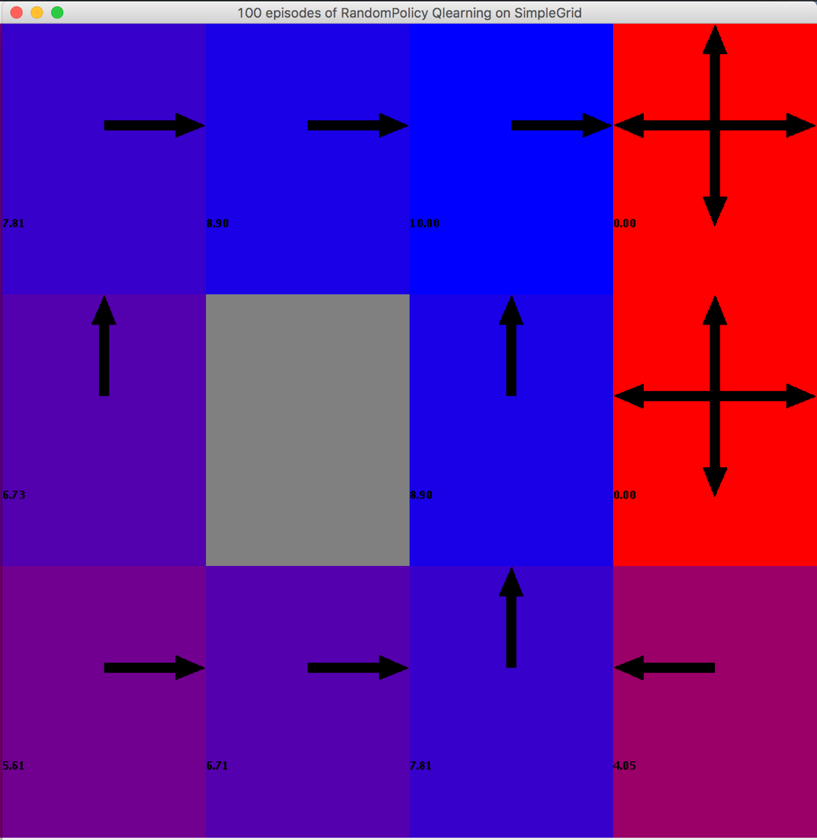
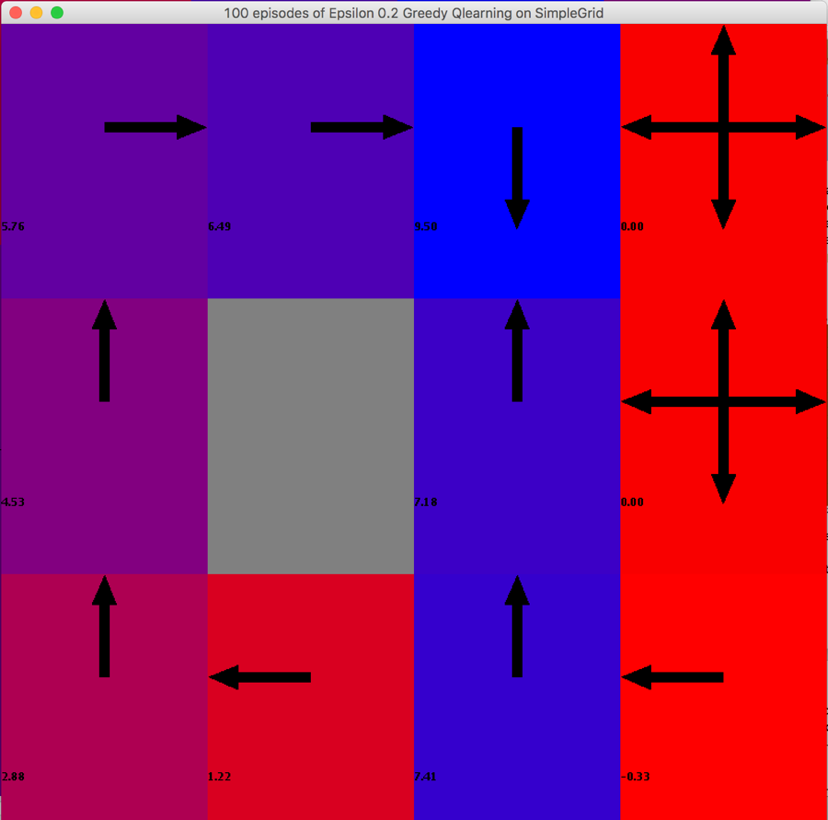
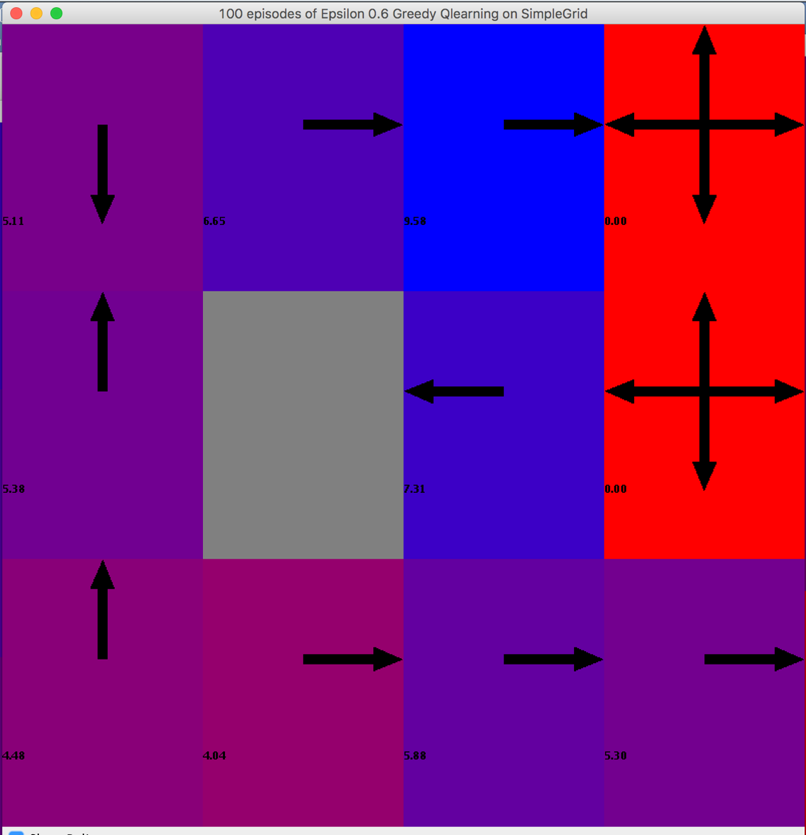
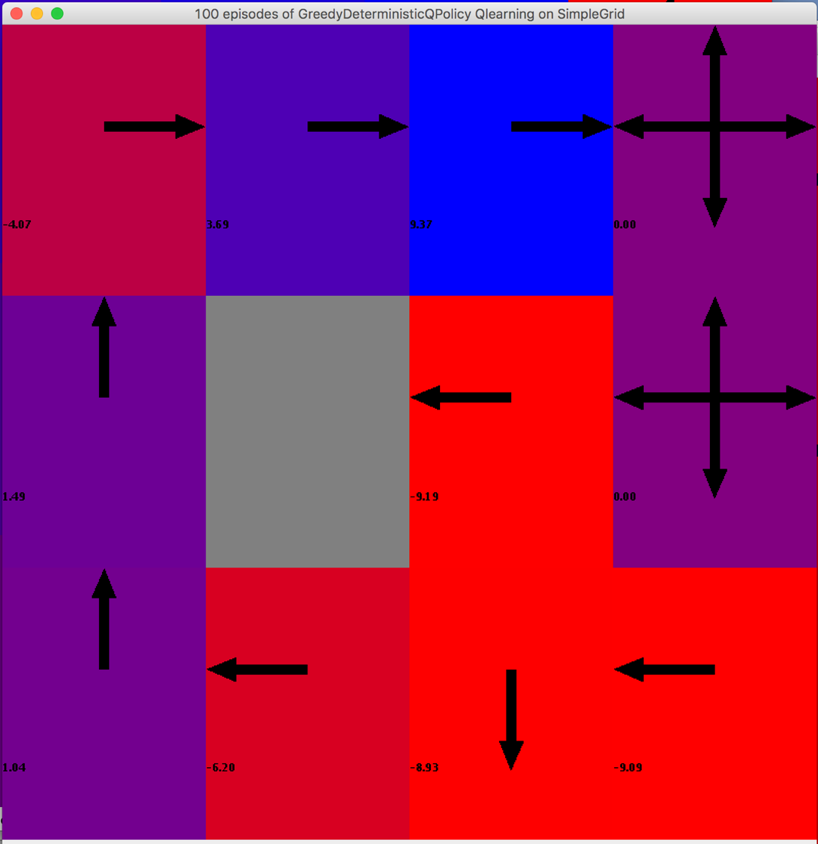
Since QLearning has no knowledge of the underlying transition model at all, it has to do some exploration, at least in the beginning, to learn the model from various experience tuples (s, a, s’, r). By exploration, it means given a state, the agent chooses a random action to move to the next state which is returned by the simulated environment. But later on, it wants to exploit the learned Q values to act well and update the Q function. QLearning goes between exploration and exploitation. To what extent it explores and exploits is determined by the different learning policies that QLearner utilizes to choose an action at each state in an episode. Here I tried four different learning policies to demonstrate how the exploration strategies can affect the behavior of QLearning.

RandomPolicy is a uniform random policy. With it, agent always chooses a random action from the action space at a step in an episode. For EpsilonGreedyPolicy, given an epsilon value, the agent will choose a random action with probability epsilon. With probability 1-epsilon the agent will choose the action with higher Q value. Thus with a higher epsilon value, agent is more likely to choose a random action over an optimal action. GreedyDeterministicQPolicy is a greedy policy that breaks ties by choosing the first action with the maximum Q value. So there is no randomness in the action selection in this policy.

|  |  |  |  |
| --- | --- | --- | --- |
| Learning Policy | Epsilon Value | Number of Episodes Trained | Time Used for Planning (ms) |
| Random | - | 100 | 36.020782 |
| EpsilonGreedy | 0.6 | 100 | 80.160604 |
| EpsilonGreedy | 0.2 | 100 | 19.444191 |
| GreedyDeterministic | - | 100 | 18.269494 |

Above table shows that with the same number of episodes used in planning, different learning policies cause the time QLearning takes to vary significantly. The only difference in the workflow they deliver a change is how long QLearning takes to choose an action at each state. From the results, we can see that with some randomness involved, QLearning takes longer to choose the action.

Below four figures display the policies after 100 episodes of planning for different learning policies. By comparing their resulting utility values and policies to the converged one, we can see that none of the learning policy trial resulted in converged solutions. Interestingly, total randomness in choosing actions when simulating this small grid problem seems to provide a quickest way to converging utilities, because in the randomPolicy graph, overall values are the closest to those of converged graph. All the state utilities are possible to be touched upon during the random process. In the GreedyDeterministic policy, the optimal path was labeled correctly but the other path to +10 terminal state was far off the ground truth solutions. This could be due to the fact that the policy always exploits from Q functions so it has always taken the optimal path, such that only optimal path is updated and the other path is ignored. The exploration and exploitation ratios of other two policies stand in between random and greedy so their behaviors also have a mixture of two. In all, small gridworld prefers a exploration-dominant approach in QLearning.

## Large Grid

This large grid shows a smoother increasing curve of rewards over number of episodes trained. As mentioned before, as more episodes are simulated and used to update Q functions, more information is learned by QLearning to provide a better optimal policy. Best policy can be found with about 150 episodes, whereas simple grid needed only 30 episodes to reach an optimal policy.

### Different Exploration Strategies

|  |  |  |  |
| --- | --- | --- | --- |
| Learning Policy | Epsilon Value | Number of Episodes Trained | Time Used for Planning (ms) |
| Random | - | 200 | 2668.131238 |
| EpsilonGreedy | 0.6 | 200 | 477.051927 |
| EpsilonGreedy | 0.2 | 200 | 154.628697 |
| GreedyDeterministic | - | 200 | 112.207487 |

Since large grid is larger, I changed the number of episodes trained to 200 just to get a more developed policy layout. This table shows more clearly that it would take longer to process if agent chooses an action randomly at a state.

Compared to small grid, this large grid prefers an exploitation-dominant strategy, in terms of how fast it converges. The greedy policy graph shows a value/policy layout that is most similar to a converged version. In the top two figures where random actions are more likely to be chosen, some states have incorrect arrows even in optimal path alone. This indicates that too much exploration in QLearning for large grid is not helpful in getting the optimal policy.

