CS 580 Reinforcement Learning
HW1
Yang Zhang 11529139

Part I. Implementation of Value Iteration and Policy Iteration

Result:

```
valueIteration, with gamma = 0.9
V*:  [75.3, 87.0, 100.0, 0, 64.77, 0.0, 87.0, 0, 55.293, 64.77, 75.3, 64.77]

policyIteration, with gamma = 0.9
PI*:  [3, 3, 3, 2, 0, 1, 0, 0, 3, 3, 0, 2]
V*:  [75.3, 87.0, 100.0, 0, 64.77, 0.0, 87.0, 0, 55.293, 64.77, 75.3, 64.77]
```

After both algorithms reached their convergence, they both end up with the same value function.

Part II. Empirically compare how long learning takes, and the policy produced, when the discount factor in the task changed

Observation : From the results below, the learning iteration times is always 5 for gamma between 0.1 to 0.9, and all the runs give out the identical optimal policy. This may be because the world in this case is quite simple, so that the learning rate doesn't affect learning significantly. However, in general, higher learning rate should cause faster learning process.

Results for gamma between 0.1 and 0.9:

```
policyIteration, with gamma = 0.1
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [-2.3, 7.0, 100.0, 0, -3.23, 0.0, 7.0, 0, -3.323, -3.23, -2.3, -3.23]


policyIteration, with gamma = 0.2
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [0.40000000000000036, 17.0, 100.0, 0, -2.92, 0.0, 17.0, 0, -3.584, -2.92, 0.40000000000000036, -
2.92]


policyIteration, with gamma = 0.3
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [5.1, 27.0, 100.0, 0, -1.4700000000000002, 0.0, 27.0, 0, -3.441, -1.4700000000000002, 5.1, -
1.4700000000000002]
```

```
policyIteration, with gamma = 0.4
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [11.8, 37.0, 100.0, 0, 1.7200000000000006, 0.0, 37.0, 0, -2.312, 1.7200000000000006, 11.8,
1.7200000000000006]


policyIteration, with gamma = 0.5
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [20.5, 47.0, 100.0, 0, 7.25, 0.0, 47.0, 0, 0.625, 7.25, 20.5, 7.25]


policyIteration, with gamma = 0.6
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [31.199999999999996, 57.0, 100.0, 0, 15.719999999999995, 0.0, 57.0, 0, 6.431999999999997,
15.719999999999995, 31.199999999999996, 15.719999999999995]


policyIteration, with gamma = 0.7
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [43.9, 67.0, 100.0, 0, 27.729999999999997, 0.0, 67.0, 0, 16.410999999999998,
27.729999999999997, 43.9, 27.729999999999997]


policyIteration, with gamma = 0.8
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [58.6, 77.0, 100.0, 0, 43.88, 0.0, 77.0, 0, 32.104000000000006, 43.88, 58.6, 43.88]


policyIteration, with gamma = 0.9
iteration: 5
PI*:  [3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2]
V*:  [75.3, 87.0, 100.0, 0, 64.77, 0.0, 87.0, 0, 55.293, 64.77, 75.3, 64.77]
```

Part III. Implement an asynchronous dynamic programming method. Modify your algorithm so that it doesn't always have to do a full sweep of the state space in the same manner each time. Does it help or hurt the performance? Why?

Result from standard value iteration algorithm:

```
valueIteration, with gamma = 0.9
V*:  [75.3, 87.0, 100.0, 0, 64.77, 0.0, 87.0, 0, 55.293, 64.77, 75.3, 64.77]
Iterations:  11
```

Result from prioritized value iteration (asynchronous DP) algorithm:

```
valueIteration, with gamma = 0.9
V*:  [-5.7, 87.0, 100.0, 0, -5.7, 0.0, 87.0, 0, -5.7, 64.77, 75.3, 64.77]
Iterations:  26
```

The standard version needs less iterations to reach its convergence. This is because the world in this case is this case is simple, so that it is fast to spread the influence of final state to other state even if doing a full sweep. In other words, full sweep method send more information to states than prioritized method with the almost same amount of time. The more information sent, the faster it converged.