Deep Reinforcement Learning (CS 545)
Assignment Report
Assignment-1: Value Iteration

Osman Furkan Kınlı – S002969

Problem definition:

In this assignment, the value iteration algorithm is implemented for a given environment in Python 3.X. In this environment, an agent is placed to a certain position in N x N grid and the goal of this agent is to reach the target cell in the grid. Only movements that the agent can make are going up, down, left and right. It cannot move cross-directional. Moreover, at the beginning of the simulation, each cell in the grid is has one attribute of being floor or being mountain. The costs of moving from a cell to another cell as follows:

- Floor TO Floor: -1
- Floor TO Mountain: -3
- Mountain TO Mountain: -2
- Mountain TO Floor -1
- Any of them TO Goal 10

Solution approach:

The main problem in this assignment is to be reached the target cell in the grid by the agent. To solve it, we use iterative value approximation algorithm for this problem. This algorithm is also called value iteration algorithm. The idea behind this algorithm is that the expected sum of rewards accumulated by starting from state *s* and acting optimally in step *i*. In other words, it computes the optimal state value function by iterative approximating the values and updating the Q until convergence. The proof of this algorithm as follows:

$$Q^*(s,a) = R(s,a) + \gamma\, \mathbb{E}_{s'}\left[V^*(s')\right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^*(s')$$

Since,

$$V^*(S) = \max_a Q^*(s,a)$$

$$V^*(S) = \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^*(s') \right]$$

Pseudo-code:

```
• Initialize the values arbitrarily (e.g. zeros)
• Repeat
    ○ delta := 0
    ○ For each state in the grid
        ▪ tmp_v = v(s)
        ▪ For each action in this state
            • Find new state after acting
            • Calculate reward after acting
            • Calculate the value of this state according to the formula
            • (new_value := reward + discount_factor * v(new_state))
            • Evaluate the new value is maximum for this state
        ▪ Calculate the amount of change for the value in this state
        ▪ Update delta with the maximum of delta and the change in the value
• until delta < small epsilon (1e-2)
```

Details:

In this assignment, there are 5 different environments to run this algorithm. The output is dumped as dictionary object of 2-dimensional value array that keeps the final value of each state after the agent reaches the target cell in the grid as value, and the name of environment as key to pickle object. The results as follows:

```
'Grid1.pkl':    array([[-1.,  2.,  3.,  4.,  5.,  2.],
                       [ 2.,  3.,  4.,  5.,  6.,  5.],
                       [ 3.,  4.,  5.,  7.,  7.,  6.],
                       [ 2.,  5.,  6.,  9.,  8.,  7.],
                       [ 5.,  6.,  9., 10.,  9.,  8.],
                       [ 6.,  7., 10.,  0., 10.,  9.]])

'Grid2.pkl':    array([[ 6.,  7.,  8.,  9., 10.,  9.,  8.],
                       [ 7.,  8.,  9., 10.,  0., 10.,  9.],
                       [ 6.,  7.,  8.,  9., 10.,  9.,  8.],
                       [ 4.,  6.,  7.,  8.,  9.,  6.,  7.],
                       [ 4.,  5.,  6.,  5.,  6.,  5.,  4.],
                       [ 3.,  4.,  5.,  4.,  5.,  2.,  3.],
                       [ 2.,  3.,  4.,  3.,  3.,  1.,  2.]])

'Grid3.pkl':    array([[ 5.,  5.,  5.,  4.,  3.,  2.,  1.],
                       [ 6.,  7.,  6.,  5.,  4.,  3.,  2.],
                       [ 7.,  8.,  7.,  6.,  5.,  4.,  3.],
                       [ 8.,  9.,  8.,  7.,  6.,  5.,  4.],
                       [ 9., 10.,  9.,  8.,  7.,  5.,  3.],
                       [10.,  0., 10.,  9.,  6.,  5.,  4.],
                       [ 9., 10.,  9.,  8.,  7.,  6.,  3.]])
```

```
'Grid4.pkl':    array([[-3., -2., -1.,  0.,  1.,  2.,  3.,  2.],
                       [-2., -1.,  0.,  1.,  2.,  3.,  4.,  3.],
                       [-1.,  0.,  1.,  2.,  3.,  4.,  5.,  4.],
                       [ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  5.],
                       [-1.,  2.,  3.,  4.,  5.,  6.,  7.,  6.],
                       [-1.,  2.,  3.,  4.,  6.,  7.,  8.,  7.],
                       [ 0.,  3.,  4.,  5.,  8., 10.,  9.,  8.],
                       [ 2.,  4.,  5.,  8., 10.,  0., 10.,  9.]])

'Grid5.pkl':    array([[ 5.,  6.,  5.,  2.,  3.,  2.,  1.,  0.],
                       [ 6.,  7.,  6.,  5.,  4.,  3.,  2.,  1.],
                       [ 7.,  8.,  7.,  6.,  5.,  4.,  3.,  2.],
                       [ 8.,  9.,  8.,  7.,  6.,  5.,  4.,  1.],
                       [ 9., 10.,  9.,  6.,  5.,  4.,  3.,  2.],
                       [10.,  0., 10.,  8.,  6.,  3.,  2.,  1.],
                       [ 9., 10.,  9.,  8.,  7.,  5.,  2.,  1.],
                       [ 8.,  9.,  8.,  7.,  6.,  5.,  2.,  1.]])
```

P.S:

All codes have been implemented from scratch.

Thanks to our TA Emir for giving clear intuition about value iteration algorithms in his office hour.