Assignment 2 AI
Team - 25

- Pranay Gupta, 20161088
- Shubh Maheshwari, 20161170

R(s, a) = reward for states all states

-25	0	0	0
0	0	0	0
2.5	0	0	0
0	-2.5	0	25

Step Reward = -5
U_t(i,j) = Utility of state i,j at time t
Gamma = 0.9
Terminal state = [(0,0),(3,3)]
Wall = [(1,2)]

Value Iteration
We run this algorithm for all states except terminal states and walls (i, j) denotes i^{th} and j^{th} column until we reach tolence

Iteration 1 :

Epoch:0 Error:15.000000

Updating:0 1 -5.000000
Updating:0 2 -5.000000
Updating:0 3 -5.000000
Updating:1 0 -5.000000
Updating:1 1 -5.000000
Updating:1 3 -5.000000
Updating:2 0 -2.500000
Updating:2 1 -5.000000
Updating:2 2 -5.000000
Updating:2 3 15.000000
Updating:3 0 -5.000000
Updating:3 1 -7.500000
Updating:3 2 15.000000

V V > V

V V - V
V < V V
^ > > ^
-25.000 -5.000 -5.000 -5.000
-5.000 -5.000 0.000 -5.000
-2.500 -5.000 -5.000 15.000
-5.000 -7.500 15.000 25.000

This is the first iteration and so basically the policy can be seen as a randomized version where there are many issues with it and clearly we won’t be able to reach the end state.

Iteration 2 :

Epoch:1 Error:13.000000

Updating:0 1 -10.000000
Updating:0 2 -10.000000
Updating:0 3 -10.000000
Updating:1 0 -8.000000
Updating:1 1 -10.000000
Updating:1 3 6.000000
Updating:2 0 -7.250000
Updating:2 1 -8.250000
Updating:2 2 8.000000
Updating:2 3 16.000000
Updating:3 0 -8.250000
Updating:3 1 3.250000
Updating:3 2 16.000000

V V > V
V **<** - V
^ > V V
> > > ^

-25.000 -10.000 -10.000 -10.000
-8.000 -10.000 0.000 6.000
-7.250 -8.250 8.000 16.000
-8.250 3.250 16.000 25.000

If we see in the first iteration the policy had a lot of problems. For example we would be stuck in this loop (see position (3, 0) and (2, 0)) of going up and down. So we would certainly need to update our policy. This is the second iteration so it's basically just making the initial policy changes which are deemed good and which would result in us reaching the terminal state from most states. But still there are some issues with this. For ex we see the position (1,0) ans (2,0). We would get stuck in this loop.

<u>Iteration 3</u> :

Epoch:2 Error:8.975000

Updating:0 1 -15.000000
Updating:0 2 -15.000000
Updating:0 3 -2.200000
Updating:1 0 -12.600000
Updating:1 1 -13.225000
Updating:1 3 9.000001
Updating:2 0 -10.450000
Updating:2 1 0.725000
Updating:2 2 10.200001
Updating:2 3 17.400000
Updating:3 0 -3.950000
Updating:3 1 4.800000
Updating:3 2 17.400000

V V > V
V **V** - V
> > V V
> > > ^

-25.000 -15.000 -15.000 -2.200
-12.600 -13.225 0.000 9.000
-10.450 0.725 10.200 17.400
-3.950 4.800 17.400 25.000

This iteration finally makes the desired changed and we see that we would reach the terminal state if we start from any position.

<u>Iteration 8</u> :

Epoch:7 Error:1.365857

Updating:0 1 -8.519598
Updating:0 2 -2.177538

Updating:0 3 4.522495
Updating:1 0 -5.295281
Updating:1 1 -1.530580
Updating:1 3 11.776512
Updating:2 0 1.342914
Updating:2 1 5.555642
Updating:2 2 12.478209
Updating:2 3 18.048256
Updating:3 0 1.786273
Updating:3 1 8.289510
Updating:3 2 18.048256

V V > V
V V - V
> > V V
> > > ^

-25.000 -8.520 -2.178 4.522
-5.295 -1.531 0.000 11.777
1.343 5.556 12.478 18.048
1.786 8.290 18.048 25.000

Iteration 13 :

Epoch:12 Error:0.047148

Updating:0 1 -7.007732
Updating:0 2 -1.492260
Updating:0 3 4.768949
Updating:1 0 -3.999558
Updating:1 1 -0.908652
Updating:1 3 11.805258
Updating:2 0 1.890977
Updating:2 1 5.741687
Updating:2 2 12.499777
Updating:2 3 18.055481
Updating:3 0 2.074980
Updating:3 1 8.353044
Updating:3 2 18.055481

V V > V
V V - V

```
> > V V
> > > <
```

-25.000 -7.008 -1.492 4.769
-4.000 -0.909 0.000 11.805
1.891 5.742 12.500 18.055
2.075 8.353 18.055 25.000

<u>Iteration 14</u> :

Updating:0 1 -8.839069
Updating:0 2 -4.432712
Updating:0 3 1.202816
Updating:1 0 -6.233100
Updating:1 1 -3.887096
Updating:1 3 8.324695
Updating:2 0 -1.138901
Updating:2 1 2.117785
Updating:2 2 9.057183
Updating:2 3 15.730927
Updating:3 0 -1.130982
Updating:3 1 4.963507
Updating:3 2 15.730927
Epoch:13 Error:0.005090

```
V V > V
V V - V
> > V V
> > > ^
```

-25.000 -8.839 -4.433 1.203
-6.233 -3.887 0.000 8.325
-1.139 2.118 9.057 15.731
-1.131 4.964 15.731 25.000

<u>Iteration 29(Final State)</u>

Updating:0 1 -6.965829
Updating:0 2 -1.475695
Updating:0 3 4.774305
Updating:1 0 -3.956245

Updating:1 1 -0.886900
Updating:1 3 11.805555
Updating:2 0 1.910087
Updating:2 1 5.746768
Updating:2 2 12.500000
Updating:2 3 18.055555
Updating:3 0 2.082969
Updating:3 1 8.354579
Updating:3 2 18.055555
Epoch:29 Error:0.000000
V V > V
V V - V
> > V V
> > > ^

-25.000 -6.966 -1.476 4.774
-3.956 -0.887 0.000 11.806
1.910 5.747 12.500 18.056
2.083 8.355 18.056 25.000

Result : We have finally reached the optimum policy state and as we can see we would reach the terminal state now if we start from any position on the grid.

Utility:

-25.000 -6.966 -1.476 4.774
-3.956 -0.887 0.000 11.806
1.910 5.747 12.500 18.056
2.083 8.355 18.056 25.000

Final Policy:

V V > V
V V - V
> > V V
> > > ^

If we start from (3,0)

We will always take right (3,0) => (3,1) => (3,2) => (3,3)

If we start from (1,0)

We will still reach (3,3) but we will take +2.5 reward and avoid -2.5 reward
(1,0)=>(2,0)=>(2,1)=>(2,2)=>(3,2)=>(3,3)

<u>Linear Programming</u>

Values of X

XXXXXXXX X	States	X
(State,Actions)		
(0, 0)	**noop**	0
(0, 1)	**North**	0
(0, 1)	**South**	0
(0, 1)	**East**	0
(0, 1)	**West**	0
(0, 2)	**North**	0
(0, 2)	**South**	0
(0, 2)	**East**	0
(0, 2)	**West**	0
(0, 3)	**North**	0
(0, 3)	**South**	0
(0, 3)	**East**	0
(0, 3)	**West**	0
(1, 0)	**North**	0
(1, 0)	**South**	0.016998796
(1, 0)	**East**	0

(1, 0)	**West**	0
(1, 1)	**North**	0
(1, 1)	**South**	0.0268777785
(1, 1)	**East**	0
(1, 1)	**West**	0
(1, 3)	**North**	0
(1, 3)	**South**	0
(1, 3)	**East**	0
(1, 3)	**West**	0
(2, 0)	**North**	0
(2, 0)	**South**	0
(2, 0)	**East**	0.1261113855
(2, 0)	**West**	0
(2, 1)	**North**	0
(2, 1)	**South**	0
(2, 1)	**East**	0.2249012102
(2, 1)	**West**	0
(2, 2)	**North**	0
(2, 2)	**South**	0
(2, 2)	**East**	0.3388010757
(2, 2)	**West**	0
(2, 3)	**North**	0
(2, 3)	**South**	0.30115651

		17
(2, 3)	**East**	0
(2, 3)	**West**	0
(3, 0)	**North**	0
(3, 0)	**South**	0
(3, 0)	**East**	1.1251234873
(3, 0)	**West**	0
(3, 1)	**North**	0
(3, 1)	**South**	0
(3, 1)	**East**	1.0250987898
(3, 1)	**West**	0
(3, 2)	**North**	0
(3, 2)	**South**	0
(3, 2)	**East**	0.9488434883
(3, 2)	**West**	0
(3, 3)	**noop**	1

Expected Reward from linear programming :

2.082968874

Expected Reward from Value Iteration :

2.083

Why is the reward similar?

We try to maximize the utility/reward in both the methods of solving the MDP, so they would both end up achieving the same result if we try make them as accurate as possible. In VI, the reward in the start state is the utility of selecting the best paths possible to the terminal states. In LP, the paths we get match the ones in VI so they will also consider similar probabilities. Now the reward in LP is the summation of the reward*x for each state,action pair. This will correspond to the value we get in VI if we assume a small delta, since a large delta would not allow enough iterations so that our VI spreads out enough and approximates the utilities of different states enough times. So if we use a delta not near 0, the values in VI and LP might not match but on using delta near 0, as in our case the rewards match in both of them.