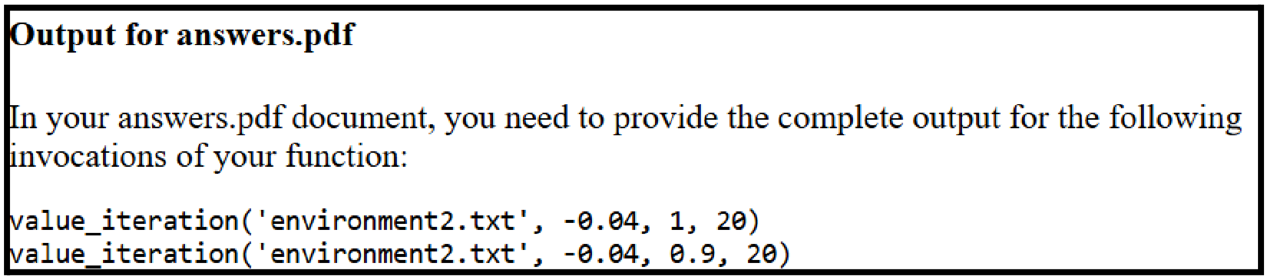
Nicholas Nhat Tran 1002027150

CSE 4309 Assignment 7

Task 1 (70 pts)

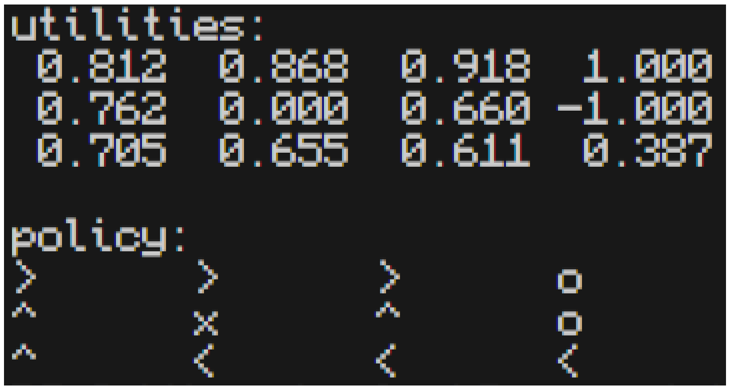


Output for answers.pdf

In your answers.pdf document, you need to provide the complete output for the following invocations of your function:

```
value_iteration('environment2.txt', -0.04, 1, 20)
value_iteration('environment2.txt', -0.04, 0.9, 20)
```

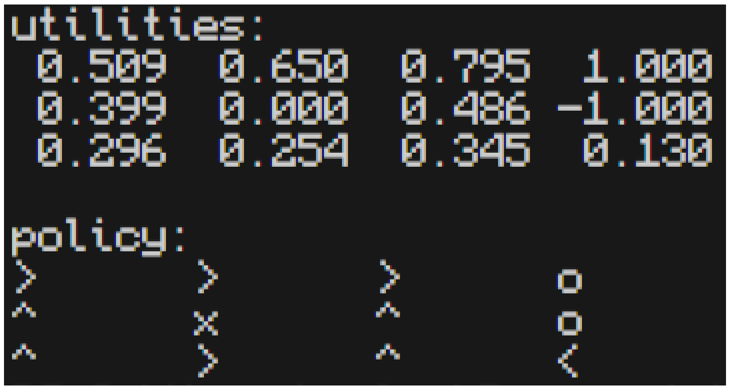
value_iteration('environment2.txt', -0.04, 1, 20)



```
utilities:
 0.812  0.868  0.918  1.000
 0.762  0.000  0.660 -1.000
 0.705  0.655  0.611  0.387

policy:
>      >      >      o
^      x      ^      o
^      <      <      <
```

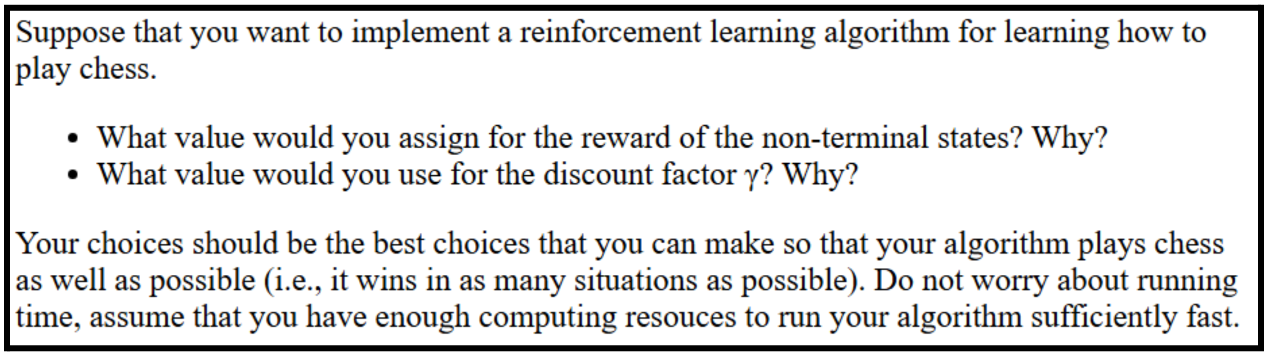
value_iteration('environment2.txt', -0.04, 0.9, 20)



```
utilities:
 0.509  0.650  0.795  1.000
 0.399  0.000  0.486 -1.000
 0.296  0.254  0.345  0.130

policy:
>      >      >      o
^      x      ^      o
^      >      ^      <
```

Task 2 (10 pts)



Suppose that you want to implement a reinforcement learning algorithm for learning how to play chess.

- What value would you assign for the reward of the non-terminal states? Why?
- What value would you use for the discount factor γ? Why?

Your choices should be the best choices that you can make so that your algorithm plays chess as well as possible (i.e., it wins in as many situations as possible). Do not worry about running time, assume that you have enough computing resouces to run your algorithm sufficiently fast.

What value would you assign for the reward of the non-terminal states? Why?

I would assign the reward of the non-terminal states as 0 because it will focus purely on the final outcome. Setting the reward to a small negative value may direct the algorithm to prioritize "speed", which may end up choosing a 50-move draw over a 50-move win.

What value would you use for the discount factor γ? Why?

I would assign the discount factor as 1, as it represents the "perfectly patient" agent. Any discount factor lower than 1 will begin prioritizing speed over longer move sequences, meaning it may choose a 5-move sequence with an 80% chance of winning over a 50-move sequence with a 90% chance of winning. This means that with a discount factor of 1, the algorithm will choose the path with the highest probability of winning every time.

Task 3 (20 pts)

Part a

U(1,2) = -1.0 (Terminal)
U(3,2) = 1.0 (Terminal)

U(2,1) = 0.0 (Blocked)
U(2,3) = 0.0 (Blocked)

Up Action (80/10/10 Intended/Slip/Slip)

E(UP) = 0.8 * U(3,2) + 0.1 * U(2,1) + 0.1 * U(2,3)
E(UP) = 0.8 * 1.0 + 0.1 * 0.0 + 0.1 * 0.0
E(UP) = 0.8

Down Action (80/10/10 Intended/Slip/Slip)

E(DOWN) = 0.8 * U(1,2) + 0.1 * U(2,1) + 0.1 * U(2,3)
E(DOWN) = 0.8 * -1.0 + 0.1 * 0.0 + 0.1 * 0.0
E(DOWN) = -0.8

Left Action (80/10/10 Intended/Slip/Slip)

E(LEFT) = 0.8 * U(2,2) + 0.1 * U(1,2) + 0.1 * U(3,2)
E(LEFT) = 0.8 * U(2,2) + 0.1 * -1.0 + 0.1 * 1.0
E(LEFT) = 0.8 * U(2,2)

Right Action (80/10/10 Intended/Slip/Slip)

E(RIGHT) = 0.8 * U(2,2) + 0.1 * U(1,2) + 0.1 * U(3,2)
E(RIGHT) = 0.8 * U(2,2)

Plug into Bellman Equation

U(2,2) = R(2,2) + γ * max[E(UP), E(DOWN), E(LEFT), E(RIGHT)]
U(2,2) = -0.04 + 0.9 * max[0.8, -0.8, 0.8 * U(2,2), 0.8 * U(2,2)]
U(2,2) = -0.04 + 0.9 * max[0.8, -0.8, 0.8 * U(2,2)]

Finding U(2,2)

Assume "UP" is optimal:
U(2,2) = -0.04 + 0.9 * 0.8 = 0.68
Assume "LEFT" or "RIGHT" is optimal:
U(2,2) = -0.04 + 0.9 * (0.8 * U(2,2)) = -0.04 + 0.72 * U(2,2)

U(2,2) - 0.72 * U(2,2) = -0.04
0.28 * U(2,2) = -0.04
U(2,2) = -0.142857143

Since 0.8 > -0.142857143, the maximum utility among the three actions must be "UP".

Plug into Bellman Equation

U(2,2) = -0.04 + 0.9 * 0.8 = 0.68

Part b

E(UP) is optimal as long as it is the maximum value. E(UP) will always be greater than E(DOWN) since 0.8 > -0.8 is always true.

For E(UP) > E(LEFT), which is 0.8 > 0.8 * U(2,2), E(UP) will be optimal as long as 1 > U(2,2). Therefore, E(UP) will not be optimal if that condition is false, meaning we want to find the condition E(LEFT) >= E(UP), U(2,2) >= 1.

When "LEFT" or "RIGHT" is optimal, the Bellman equation becomes:

U(2,2) = r + 0.9 * E(LEFT) = r + 0.9 * (0.8 * U(2,2))
U(2,2) = r + 0.72 * U(2,2)
0.28 * U(2,2) = r
U(2,2) = r/0.28

We then plug this into the equality where "UP" is not optimal:

U(2,2) >= 1
r/0.28 >= 1
r >= 0.28

This means that whenever r is greater than or equal to 0.28, the "UP" action is non-optimal.