Jacob Devasier

Task 1

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

```
 0.811, 0.868, 0.918, 1.000
 0.761, 0.000, 0.660,-1.000
 0.704, 0.653, 0.608, 0.382
```

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

```
 0.509, 0.650, 0.795, 1.000
 0.398, 0.000, 0.486,-1.000
 0.296, 0.254, 0.345, 0.130
```

Task 2

q_learning('environment2.txt', -0.04, 1, 1000, 20)

```
 0.830, 0.920, 0.960, 1.000
 0.462, 0.000, 0.843,-1.000
 0.114, 0.290, 0.635, 0.417
```

q_learning('environment2.txt', -0.04, 0.9, 1000, 20)

```
 0.477, 0.701, 0.860, 1.000
 0.240, 0.000, 0.700,-1.000
 0.054, 0.288, 0.531, 0.328
```

Task 3

The reward for any non-terminal state would be the sum of pieces for each team where white pieces are positive and black pieces are negative. For example, if white were up 1 pawn, the score would be +1. If black were up 2 pawns, it would be -2, and so on.

The value for gamma would be 1 because the number of moves doesn’t matter as long as a win is achieved (Assuming no game rules about the length of the game).

Task 4

Part A

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \left(\sum_{s'} p(s'|s,a) U(s') \right)$$

$$U(s) = -0.04 + 0.9 * \max_{a \in A(s)} \left(\sum_{s'} p(s'|s,a) U(s') \right)$$

If a = “up”

$up = p\big((3,2)\big|(2,2), up\big)U\big((3,2)\big) + \ p\big((2,2)\big|(2,2), up\big)U\big((2,2)\big)$

$up = 0.8 + 0.2 * U\big((2,2)\big)$

If a = “down”

$down = p\big((1,2)\big|(2,2), down\big)U\big((1,2)\big) + \ p\big((2,2)\big|(2,2), down\big)U\big((2,2)\big)$

$down = -0.8 + 0.2 * U\big((2,2)\big)$

If a = “left”

$left = p\big((1,2)\big|(2,2), left\big)U\big((1,2)\big) + \ p\big((2,2)\big|(2,2), left\big)U\big((2,2)\big) + p\big((3,2)\big|(2,2), left\big)U\big((3,2)\big)$

$left = 0.8 * U\big((2,2)\big)$

If a = “right”

$right = p\big((1,2)\big|(2,2), right\big)U\big((1,2)\big) + \ p\big((2,2)\big|(2,2), right\big)U\big((2,2)\big) + p\big((3,2)\big|(2,2), right\big)U\big((3,2)\big)$

$right = 0.8 * U\big((2,2)\big)$

“up” is the best choice because its utility value is at least 0.8, “down” gives a utility value of at most -0.8, and “left” and “right” give the same utility value which is always 20 percent less than the real utility value of (2,2).

Iteration 1:

$$U((2,2)) = -0.04 + 0.9 * \Big(0.8 + 0.2 * U\big((2,2)\big)\Big)$$

$$U((2,2)) = 0.68 + 0.18 * U\big((2,2)\big)$$

Iteration 2:

$$U((2,2)) = 0.68 + 0.18 * \big(0.68 + 0.18 * U((2,2))\big)$$

$$U((2,2)) = 0.8024 + 0.0324 * U((2,2))$$

Iteration 3:

$$U((2,2)) = 0.8024 + \ 0.026 + 0.00105 * U((2,2))$$

$$U((2,2)) = 0.8284 + 0.00105 * U((2,2))$$

Iteration 4:

$$U((2,2)) = 0.8284 + 0.00105 * (0.8284 + 0.00105 * U((2,2)))$$

$$U((2,2)) = 0.8284 + \ 0.001 + \cdots$$

$$\boldsymbol{U((2,2)) \approx 0.8294}$$

After running this several times, it will converge towards the real utility value. Using the program from task 1, I found that the utility value of (2,2) is **0.83** which is about what I got after 4 iterations of the recursive calculation of the expected utility function.

Part B

“up” will no longer be optimal when it is better to risk staying in the current state instead of immediately going to the terminal state. This means that we are looking for a reward that gives “left” or “right” a better utility than “up”.

Using the equation from part (a):

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \left(\sum_{s'} p(s'|s,a) U(s') \right)$$

We know that the best action will be either left or right. They will both be equal because they have the same probability of going to each state.

$$\begin{aligned} U(s) = R(s) + 0.9\, \big[\, & p((1,2)|(2,2), left) U((1,2)) + \ p((2,2)|(2,2), left) U((2,2)) \\ & + p((3,2)|(2,2), left) U((3,2))\big] \end{aligned}$$

$$U(s) = R(s) + 0.9\, [\, 0.1 * \ -1 + \ 0.8 * \ U((2,2)) + 0.1 * 1]$$

$$U(s) = R(s) + 0.9 * \ \big(0.8 * \ U((2,2))\big)$$

After 4 recursions:

$$U(s) = R(s) + 0.9(0.8[R(s) + 0.9 * (0.8 * [R(s) + 0.9 * \ (0.8 * [R(s) + 0.9 * (0.8 * \ldots)])])])$$

Now we need to find a value for R(s) that satisfies U(2,2) > 1 so that the algorithm will want to go left instead of up. When plugging the inequality into Wolfram Alpha, I got *r > 0.3829*. This gives us the lower bound and because we can plug in any number greater than *0.3829*, it tells us that there is no upper bound.

r = (0.3829, infinity)