Task 1

Code completed in value\_iteration.py

value\_iteration.py '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 0.388

policy:

> > > o

^ x ^ o

^ < < <

value\_iteration.py '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

I would have three terminal states:

* Checkmate (win): 1
* Stalemate (tie): 0
* Checkmated (lose): -1

I’ve made these decisions because chess only has three outcomes which corelate to the terminal states above. Because we are trying to play chess ‘well’ we should give a positive reward when winning and a negative reward when losing. Because stalemates also exist, I’ve set its reward to 0 because it is neither a win nor a loss. This way a win is preferred over a stalemate and a stalemate is preferred over a loss.

I would set the discount factor to be a value less than 1. This is because you do not know your opponent’s move before they take it. In this way chess is non-deterministic and we can’t know how the game behaves non-deterministically (probability of opponent moves aren’t defined). Because we can only guess our opponent’s move, future states have a high chance to deviate from the predicted state, the discount factor should be relatively low; a value of 0.5 might be a good value. This way future rewards are discounted exponentially so we don’t depend on the opponent making certain moves.

Task 3

1. I’ll first make some assumptions to avoid a majority of the calculations: ‘up’ is the optimal utility. I’m assuming this because ‘left’, ‘right’, and ‘down’ results has a negative utility with the given constants.

Policy: ‘up’, nts = -0.04, y = 0.9

Resulting states and probabilities:

* (2,2) = ‘left’ (hits wall)(0.1) + ‘right’ (hits wall)(0.1) = 0.2
* (3,2) = ‘up’ (0.8) = 0.8

U(2,2) = 0.8\*1 + .2\*X = 0.8 + 0.2\*(nts + y\*U(2,2))

= 0.8 - 0.008 + 0.18\*U(2,2)

.82\*U(2,2) = .792; **U(2,2) = 0.96585365**

Task 3

1. I’ll make another assumption for this part: The only relevant policies are ‘up’ and ‘left’. I’ assuming this because ‘down’ is has the same calculations as ‘up’ but with a negative terminal, so it will always have a lower utility than ‘up’. Additionally, ‘left’ and ‘right’ are symmetric given the probability distribution of [.1,.8,.1] So I will only focus on ‘left’ because their utilities are the same.

Policy: ‘up’, nts = unknown, y = 0.9

Resulting states and probabilities:

* (2,2) = ‘left’ (hits wall)(0.1) + ‘right’ (hits wall)(0.1) = 0.2
* (3,2) = ‘up’ (0.8) = 0.8

U(2,2) = 0.8\*1 + .2\*X = 0.8 + 0.2\*(nts + y\*U(2,2))

= 0.8 – 0.2\*nts + 0.18\*U(2,2)

.82\*U(2,2) = 0.8 – 0.2\*nts; U(2,2) = (0.8 – 0.2\*nts)/0.82

Policy: ‘left’, nts = unknown, y = 0.9

Resulting states and probabilities:

* (3,2) = ‘up’ (0.1) = 0.1
* (1,2) = ‘down’ (0.1) = 0.1
* (2,2) = ‘left’ (hits wall)(0.8) = 0.8

U(2,2) = 0.8\*X + 0.1\*1 + 0.1\*(-1) = 0.8(nts + y\*U(2,2)) + 0

= 0.8\*nts + 0.18\*U(2,2)

.82\*U(2,2) = 0.8\*nts; U(2,2) = (0.8\*nts)/0.82

Set calculated utility values to each other to find cross-over point.

(0.8\*nts)/0.82 = (0.8 – 0.2\*nts)/0.82

0.8\*nts = 0.8 – 0.2\*nts

**nts = 0.8 aka r = 0.8**

Ranges:

‘up’: (-∞, 0.8)

‘left’ or ‘right’: **(0.8, ∞) <- ‘up’ is not optimal.**

At r = 0.8 ‘up’ ‘left’ and ‘right’ have the same utility and are all optimal(ish)

Task 4