Kevin Jasieniecki
CS 440
12/13/12

I implemented this MP with two classes, *node* and *maze,* and a main file. I constructed the initial maze as a 2d vector. Then, I began work with the first part. For this part, I used value iteration. For some n (in my main, I use 500) times, I iterate through the 2d vector I created that represents the maze. For every possible route an agent could take at that location, I calculated the relative utility, using the formula given - .8 chance to go the intended direction, with .1 chance to go left or right. I also factored in the constant -.04 reward and the discount, as given in lecture. This iteration would ignore walls, edges of the maze, and the terminal +1 and -1 states. Once I had the utilities of each direction, I calculated the maximum of those, and then assigned the node's current direction to that maximum direction. Since the utility values per node were persistent, the direction value could change on further iterations. Finally, once all of the iterations were done, I had a fairly accurate path a starting agent could take to get to maximize its reward (attached).

I took a similar route for the TD learning section. After initializing an alpha, I chose a random direction and cell in the same maze. Essentially, once a cell next to one of the constant +1 or -1 cells was randomly chosen, the utility would be updated – cells close to the +1 would in general have a positive utility, and cell close to the -1 cell would have a negative utility. As more cells were randomly chosen, and as the function iterates (I iterated it 50000000 times), these utilities cascade out from the constant +1 and -1 cells, eventually reaching the (0,0) cell.

Things of interest to note: The directional model for the value iteration conformed very well with what would be expected of a utility-maximizing agent, while the TD-learning model was mostly correct, but different, especially around the -1 cell. Also, it took me a couple of iterations to even mark a utility at the cell immediately to the right of the (0,0) cell – since it is next to a wall on 3 sides, it was hardly ever accessed, since once needs to “chain” utilities around the walls, to the start state. It was a very tricky spot to find for a random searcher.

Attached:

For the Value iteration model (part 1)

Utilities (at 500 iterations):

```
[ 0.301 ]    [ 0.247 ]    [ 0 ]        [ 0.845 ]    [ 0.915 ]    [ 1 ]

[ 0.362 ]    [ 0 ]        [ 0.706 ]    [ 0.772 ]    [ 0.735 ]    [ -1 ]

[ 0.417 ]    [ 0 ]        [ 0.654 ]    [ 0.706 ]    [ 0 ]        [ 0.457 ]

[ 0.473 ]    [ 0.536 ]    [ 0.594 ]    [ 0.635 ]    [ 0.577 ]    [ 0.513 ]
```

Directional Model (at 500 iterations):

```
[ v ][ < ][ W ][ > ][ > ][ + ]

[ v ][ W ][ > ][ ^ ][ < ][ - ]

[ v ][ W ][ ^ ][ ^ ][ W ][ v ]

[ > ][ > ][ ^ ][ ^ ][ < ][ < ]
```

For TD Q-learning (at 5000000 iterations):

```
[ -0.0379 ]   [ -.0022 ]    [ 0 ]         [ 0.919 ]    [ 0.96 ]     [ 1 ]

[ 0.0201 ]    [ 0 ]         [ 0.77 ]      [ 0.872 ]    [ 0.911 ]    [ -1 ]

[ -0.0216 ]   [ 0 ]         [ 0.484 ]     [ 0.775 ]    [ 0 ]        [ 0.0697 ]

[ 0.0275 ]    [ 0.145 ]    [ 0.338 ]    [ 0.621 ]    [ 0.288 ]    [ 0.182 ]
```

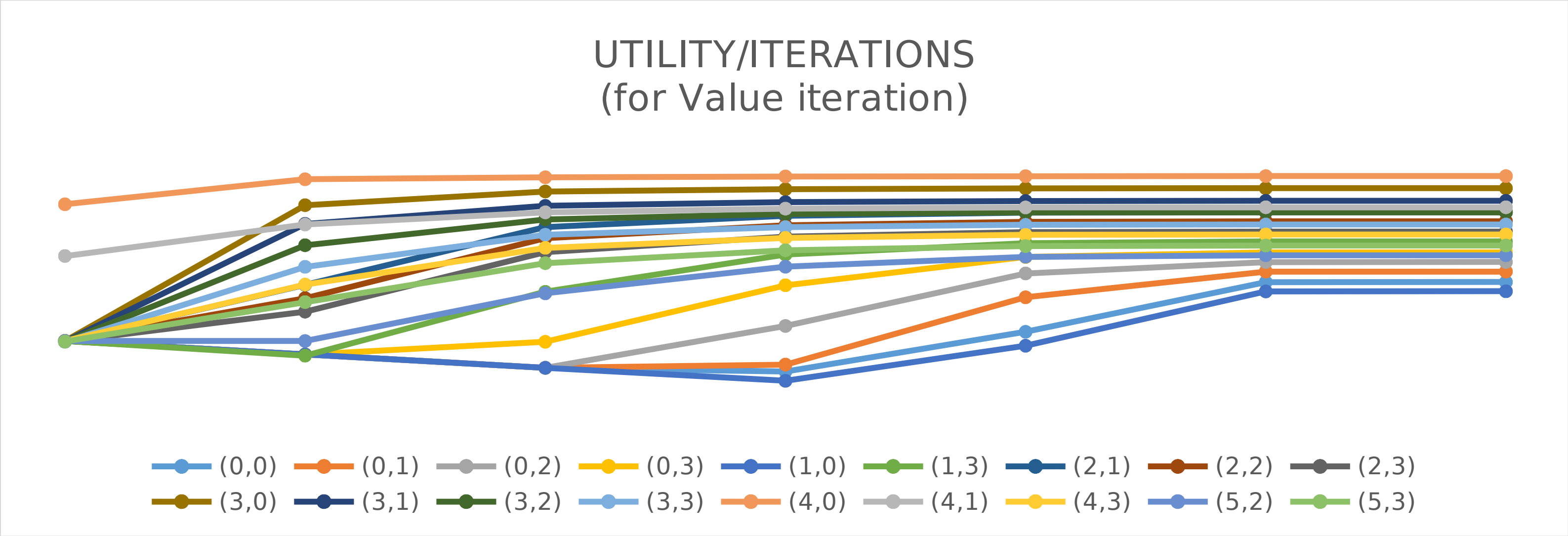
Directional Model (at 5000000 iterations):

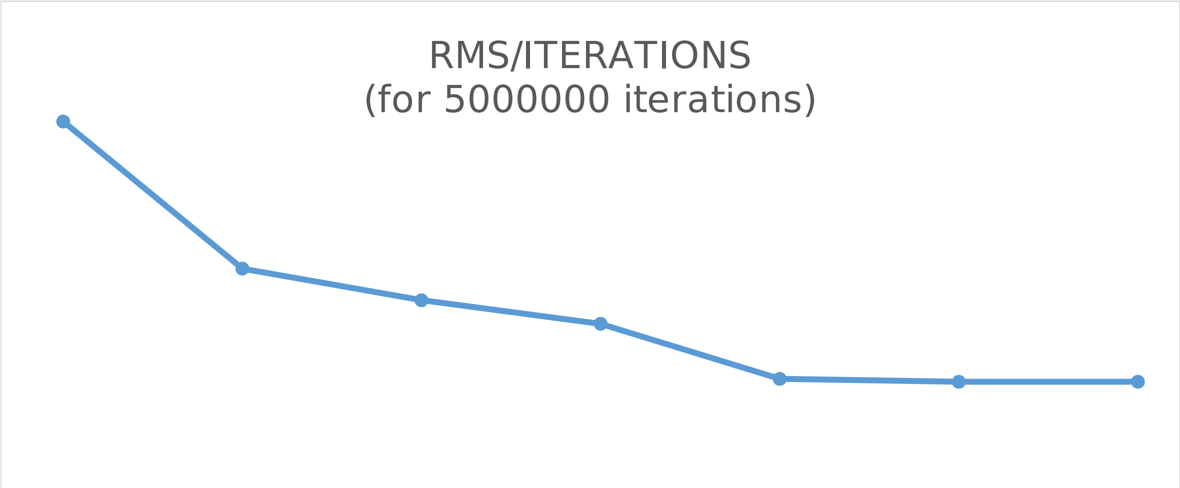
```
[ > ][ > ][ W ][ > ][ > ][ + ]

[ > ][ W ][ > ][ ^ ][ ^ ][ - ]
```

[ ^ ][ W ][ ^ ][ ^ ][ W ][ v ]

[ > ][ > ][ > ][ ^ ][ < ][ < ]

Relevant Plots:


UTILITY/ITERATIONS
(for Value iteration)
(0,0)
(0,1)
(0,2)
(0,3)
(1,0)
(1,3)
(2,1)
(2,2)
(2,3)
(3,0)
(3,1)
(3,2)
(3,3)
(4,0)
(4,1)
(4,3)
(5,2)
(5,3)


RMS/ITERATIONS
(for 5000000 iterations)