Soham Shah
Intro to AI
Rhonda Hoenigman

Assignment 7

Purpose:

The purpose of this assignment was to test how the results of the value iteration and policy iteration algorithms can be changed by changing the weights of the Markov decision matrix, changing the gamma, and adding a possible move.

Experiment 1: Changing weights

Procedure:

I created variables in the code in order to quickly change the weights of each item in the MDP.

```
nothing = -1
barn = 2
mountain = -1
snake = -0.5
wall = None
apple = 50
```

Nothing represents the non-terminal states that are not mountains, barns, or snakes and this is what we chose to experiment with. I then wrote a loop to iterate through every possible value between -1 and 1 with increments of 0.1.

```
for i in range(-10,10,1):
    nothing = i/10.0
```

I also wrote a function to compare the results of two different policy iteration outputs.

```
def compareMDP(a,b):
   for firstIndex,item1 in enumerate(a):
      for secondIndex,item2 in enumerate(item1):
         if (item2!=b[firstIndex][secondIndex]):
            return False
   return True
```

Then, it was simple to run the MDP’s for each given value in order to figure out which MDP’s differed from when the weights were 0.

Data:

Here's the directions that policy iteration gives when the non-terminal nodes are weighted at 0:

```
['>', '>', '>', '>', '>', '>', '>', '>', '>', '>']
[None, None, '>', '>', '>', '^', None, '^', None, '^']
['>', '>', '>', '>', '^', '^', None, '^', '>', '^']
[None, '^', None, None, None, '>', '>', '^', None, '^']
[None, '^', '<', '<', '<', None, '^', '^', '>', '^']
['>', '^', None, 'v', 'v', None, '^', '^', None, '^']
['^', '^', None, 'v', 'v', None, '^', '^', None, None]
['>', '>', '>', '>', '>', '>', '^', '^', '<', '<']
```

Here's the results when the non-terminal nodes are weighted at -1:

```
['>', '>', '>', '>', '>', '>', '>', '>', '>', '>']
[None, None, '>', '>', '>', '^', None, '^', None, '^']
['>', '>', '>', '>', '>', '^', None, '^', '>', '^']
[None, '^', None, None, None, '>', '>', '^', None, '^']
[None, '^', '<', '<', '<', None, '^', '^', '>', '^']
['>', '^', None, 'v', 'v', None, '^', '^', None, '^']
['^', '^', None, 'v', 'v', None, '^', '^', None, None]
['>', '>', '>', '>', '>', '>', '^', '^', '<', '<']
```

Note the node in the third list from the top and 5 over from the left. This node changes to a right movement instead of up. The same is true for all weights -1 to -0.6

Results:

The finding was that the movements change when the non-terminal nodes are 0 and when they're -1 to -0.6. This makes intuitive sense. As the snakes are weighted at -0.5, they are less painful than the non-terminal empty nodes. So the algorithm actually has the horse go over the snakes instead of the empty nodes. Also, all of the values still result in the horse being able to find the apple. This is because it is worth 50 which still outweighs the number of things the horse needs to do to get there.

Experiment 2: Changing gamma

Procedure:

Look through different values of gamma in order to see if the results differ from the standard gamma of 0.9

In order to do this, I needed to change the __init__ for the class GridMDP from:

```
def __init__(self, grid, terminals, init=(0, 0), gamma = 0.9):
```

In the init for the MDP, I essentially overrode this gamma value by putting,

GridMDP([array of weights], terminals=[(9,9)], gamma = gam, actlist=actlist) where gam different values. That gamma value is then used in the algorithm in order to figure out the values in the value iteration algorithm by weighting the probability of each potential action. The same is true for the policy evaluation algorithm.

Then, I wrote a for loop to generate possible values of gamma and see what worked.

```
for i in range(1,10,1):
    gamma = i/10.0
```

I used the compare function from above in order to see if anything changed.

Data:

For each value of gamma, a different result was created from the original where gamma = 0.9
I've decided to compare two extremes where gamma was 0.1 and gamma was 0.9.

Here's the original policy iteration result where gamma = 0.9

```
['>', '>', '>', '>', '>', '>', '>', '>', '>', '>']
[None, None, '>', '>', '>', '^', None, '^', None, '^']
['>', '>', '>', '>', '^', '^', None, '^', '>', '^']
[None, '^', None, None, None, '>', '>', '^', None, '^']
[None, '^', '<', '<', '<', None, '^', '^', '>', '^']
['>', '^', None, 'v', 'v', None, '^', '^', None, '^']
['^', '^', None, 'v', 'v', None, '^', '^', None, None]
['>', '>', '>', '>', '>', '>', '^', '^', '<', '<']
```

Here is the result where gamma = 0.1

```
['>', '<', '^', '<', 'v', '^', '<', '>', '>', '>']
[None, None, 'v', '>', 'v', '<', None, 'v', None, '^']
['>', 'v', '<', 'v', '<', '<', None, 'v', '>', '^']
```

```
[None, '>', None, None, None, '>', '>', '>', None, '^']
[None, '^', '<', '<', '<', None, '^', '^', '>', '>']
['<', '^', None, '^', '^', None, 'v', '<', None, '^']
['<', '<', None, '<', '<', None, '<', '<', None, None]
['>', 'v', '>', '^', 'v', '<', 'v', 'v', '<', '<']
```

This is a really interesting result. A lot of the edges are telling us to run into the wall infinitely instead of heading in the optimal direction.

Results:

Each value from 0.1 – 0.8 in increments of 0.1 change the results of the program in some way. I decided to go closer to 0.9, my finding was that the results don't change if gamma remains within the range of 0.852 and 0.920. This means that gamma can effectively be close to 0.9 and the results will remain similar. This makes sense as gamma is used to evaluate whether to take a long term gain or a short term gain. Essentially, the closer a gamma is to 1 the more the apple reward is prioritized over a short term loss. If it's a lower gamma, that large reward is lost because it's further away and so the horse prioritized the short term benefit. In this case, sometimes, the horse will head towards the barn over the apple.

Experiment 3: Adding Movements

Procedure:

I changed the transition model in order to take into account the horse's ability to jump. In order to do this, I first changed the list of actions that are possible. In the init for the gridMDP, I required an actlist

```
def __init__(self, grid, terminals, gamma, actlist, init=(0, 0) ):
```

Then, I wrote a createMDP implementation which could keep track of actlist.

```
def createMDP(nothing = 0, barn = 2, mountain = -1, snake = -0.5, wall = None, apple =
50, gam = 0.9, actlist=orientations):
```

Now, when creating an actlist I can override it with my own list of actions I want to be able to do.

```
testMDP_Map = createMDP(actlist = [(2,0), (0, 2), (-2, 0), (0, -2)])
```

The next steps outline how I changed the transition model. As we wanted different probabilities for the transitions, I created a global variable JUMP_VALUE which tracks which transition model was being used. Then, I could modify JUMP_VALUE and check it during the transition model.

```
def T(self, state, action):
    if action == None:
        return [(0.0, state)]

    elif (JUMP_VALUE != 1):
        return [(0.5, self.go(state, action)),
                (0.5, self.go(state, (0,0)))]

    else:
        return [(0.8, self.go(state, action)),
                (0.1, self.go(state, turn_right(action))),
                (0.1, self.go(state, turn_left(action)))]
```

Now, the action given in orientations will take place 50% of the time or it will go nowhere 50% of the time.

Finally, I needed to change the to_arrows function as this was written with 1's and set it to run with JUMP_VALUE.

```
def to_arrows(self, policy):
```

```
chars = {(JUMP_VALUE, 0):'>', (0, JUMP_VALUE):'^', (-JUMP_VALUE, 0):'<', (0, -
JUMP_VALUE):'v', None: '.'}
```

Data:

Here’s the probability and direction table from the original code:

```
{(7, 3): 204.26487888804954, (4, 7): 222.78539252072008, (1, 3): 92.1671499347865, (3, 0):
81.20429739545884, (8, 0): 115.76145955821009, (7, 7): 341.56754069939353, (0, 7):
129.30826780486188, (2, 6): 149.92080699623668, (9, 4): 300.9694830482098, (3, 7):
193.24190841023787, (2, 5): 136.30966671843245, (8, 5): 300.9694830482098, (7, 2):
177.38353652879098, (4, 0): 93.61186068303209, (1, 2): 79.16615593678297, (9, 0):
101.64408079797593, (6, 7): 298.69332277560045, (3, 3): 70.35954200626053, (7, 6):
299.91283497072243, (6, 3): 179.32493724891262, (1, 5): 118.36972706115777, (3, 6):
171.608429621877, (5, 7): 258.4167292826118, (4, 1): 80.10539629655774, (1, 1):
68.12786697274046, (9, 7): 450.5484042737411, (6, 4): 202.50272003355138, (5, 6):
223.33484307017065, (0, 0): 56.00673502427972, (3, 2): 63.997301427338634, (5, 0):
108.30196136449995, (2, 7): 167.7219143488589, (7, 1): 153.22249238541824, (4, 5):
172.6069930941072, (9, 3): 260.6432162908328, (6, 0): 123.34404575897346, (1, 4):
106.37329128836907, (7, 5): 267.0597692589979, (0, 5): 103.93426689812516, (8, 7):
395.6033493286862, (9, 5): 342.7709454543098, (5, 4): 178.78262530300833, (4, 2):
69.70938223561659, (1, 0): 63.15194750192282, (9, 6): 395.6033493286862, (3, 5):
153.54031765511553, (0, 1): 61.075903314234274, (8, 3): 227.6378184168643, (7, 0):
131.8395853684767, (4, 6): 196.38629708828637, (9, 2): 228.85733061198627, (5, 5):
193.22594699057316, (6, 1): 141.29362663968965, (3, 1): 72.17195072950506, (7, 4):
233.52546598013973, (2, 0): 71.30120670385293, (6, 2): 159.42677854861242, (4, 3):
62.56330771224403, (1, 7): 147.26789476049686, (2, 3): 80.92712600667721, (0, 2):
68.67731752219102}
```

and the directions based off this data:

```
['>', '>', '>', '>', '>', '>', '>', '>', '>', '>']
[None, None, '>', '>', '>', '^', None, '^', None, '^']
['>', '>', '>', '>', '^', '^', None, '^', '>', '^']
[None, '^', None, None, None, '>', '>', '^', None, '^']
[None, '^', '<', '<', '<', None, '^', '^', '>', '^']
['>', '^', None, 'v', 'v', None, '^', '^', None, '^']
['^', '^', None, 'v', 'v', None, '^', '^', None, None]
['>', '>', '>', '>', '>', '>', '^', '^', '<', '<']
```

After changing the transition model, here is the new probabilities:

```
{(7, 3): 272.9440946653781, (4, 7): -1.8181818181818181, (1, 3): 149.4933187677375, (3, 0):
10.345562434248484, (8, 0): 0.0, (7, 7): 407.2716679185712, (0, 7): 0.0, (2, 6): -
1.8181818181818181, (9, 4): 8.999978812916881, (3, 7): 272.6360555970084, (2, 5): 0.0, (8, 5):
0.0, (7, 2): 8.999978812916881, (4, 0): 0.0, (1, 2): 15.454503080379215, (9, 0):
6.024772201346632, (6, 7): -1.8181818181818181, (3, 3): 182.71429168371387, (7, 6):
8.999978812916881, (6, 3): -1.8181818181818181, (1, 5): 182.71429168371387, (3, 6):
3.4627887302722513, (5, 7): 333.22208114171167, (4, 1): -1.8181818181818181, (1, 1):
121.4034318364841, (9, 7): 499.99894064584396, (6, 4): 0.0, (5, 6): 6.454524267462334, (0, 0):
0.0, (3, 2): 12.644585725007312, (5, 0): 8.46454337817308, (2, 7): 0.0, (7, 1):
221.49952120728096, (4, 5): 0.0, (9, 3): 334.7096844474968, (6, 0): 0.0, (1, 4):
19.999957625833762, (7, 5): 334.7096844474968, (0, 5): 0.0, (8, 7): 0.0, (9, 5):
409.089849736753, (5, 4): 8.090887903825971, (4, 2): 0.0, (1, 0): 12.644585725007312, (9, 6):
7.363615176553243, (3, 5): 223.31770302546278, (0, 1): 0.0, (8, 3): -1.8181818181818181, (7,
0): 7.363615176553243, (4, 6): 0.0, (9, 2): 7.363615176553243, (5, 5): 272.9440946653781, (6,
1): 0.0, (3, 1): 149.4933187677375, (7, 4): 10.999978812916881, (2, 0): 0.0, (6, 2): 0.0, (4, 3):
0.0, (1, 7): 223.06567106043306, (2, 3): 0.0, (0, 2): 0.0}
```

And here are the direction arrows.

```
['>', '>', '^', '>', '<', '>', '>', '>', '>', '>']
[None, None, '>', '>', '>', '>', None, 'v', None, '<']
['>', '>', '>', '>', '>', '>', None, '>', '>', '^']
[None, '>', None, None, None, '>', '>', '>', None, '<']
[None, '>', '>', '^', '^', None, '<', '>', '^', '^']
['>', '^', None, '<', '>', None, '>', '^', None, '^']
['>', '>', None, '^', '>', None, '>', '^', None, None]
['>', '^', '>', '^', '>', '<', '>', '^', '>', '^']
```

Results:

The result is that for a lot of the nodes, the weights have become a lot lower. This is because the ability to jump means that the horse can avoid ever having to go on those nodes in the first place. Instead, the horse can just jump over the obstacles and then go around them to the most efficient route. We see this in the directions as well. Because the horse can jump, there are areas on the bottom row when the horse is being told to turn back, away from the apple because there's a more efficient route for its jumping. Also, because the horse can only jump by 2 in any direction, based on the starting point of the horse, there's locations where the horse will never visit.