HMM Assignment Report

Paolo Takagi-Atilano

Introduction

This problem works much like blind robot problem in the mazeworld assignment in that you do not know where the robot is in the maze. But there are a couple key additions. First is that each non-wall tile has a color associated with it. Next, the robot has a sensor that allows it to see the tile that it is standing on, with a `0.88` probability of the robot reading the color of the tile correctly. The odds of the robot reading the color of the tile incorrectly is divded up evenly amoungst all the colors that the tile is not. The goal of this assignment is to, when inputted a sequence of colors that the robot has seen, output a probability distribution corresponding to the possible locations of the robot during each time step of the sequence. I utilized the equations provided in page 579 of the textbook for my math.

Design

`HMM.py`

Constants:

`PROB_SUCCESS` - Probability of sensor reading color correctly.

`PROB_NORTH` - Probability of robot moving North.

`PROB_EAST` - Probability of robot moving East.

`PROB_WEST` - Probability of robot moving West.

`PROB_SOUTH` - Probability of robot moving South.

HMM Class:

Data:

`maze` - Corresponding `ColorMaze` object.

`colors` - List of colors in `ColorMaze`. Kept as list to add a consistent ordering to them.

`prob_fail` - Probability of sensor reading incorrect color from tile.

`state_count` - Number of tiles in maze, including walls.

`sensor_matrices` - Dictionary of each color mapping to their corresponding sensor matrix.

`transposed_transition_matrix` - Transposed transition matrix corresponding to provided `ColorMaze`.

Functions:

`__init__(self, maze_filename)` - Constructor, initializes everything.

`set_sensor_matrices(self)` - Sets the `sensor_matrices` dictionary to every corresponding color found in the map.

`set_transition_matrix(self)` - Returns the transition matrix for the corresponding maze.

`get_initial_distribution(self)` - Returns the initial probability distribution vector.

`forward(self, sequence, output_filename)` - The forward filtering algorithm.

`index(self, x, y)` - Given maze x and y values, returns corresponding location in matrix column.

`write_distribution(self, matrix)` - Converts vector to matrix corresponding with that of the maze.

`ColorMaze.py`

I refactored the `Maze.py` file that was provided in the previous assignment. First, I deleted all of the code about robot locations and such, because that is not a part of this problem, as we are dealing with belief states about where the robot could be. Next, I added the colors into the maze, such that every tile that was not a wall (or a `#`) had a corresponding color associated with it. Finally, I changed the indexing such that `(0, 0)` correspoded with the top left of the maze rather than the bottom left. The purpose of this as to make maze coordinates consistent with matrix coordinates used in numpy.

Initial Distribution

The initial probability distribution is a vector in which each vector component corresponds to a location in the maze. Since the robot is equally likely to be in each non-wall space, the vector works by first assigning each component a value of zero. Then it counts the number of non-walls in the maze (let `n` be the number of non-wall tiles in the maze), and then it assigns each of the vector components corresponding to a non-wall tile in the maze a value of `1/n`. Below is the code:

```
# returns initial probability distribution vector
def get_initial_distribution(self):
    count = 0
    locs = set()

    # iterate through maze
    for y in range(self.maze.width):
        for x in range(self.maze.height):
            # count all non-maze tiles, remember their location
            if self.maze.get_tile(x, y) is not None:
                count += 1
                locs.add((x, y))

    # for all non-maze locations, give corresponding probability
    initial_distribution = numpy.zeros((self.maze.height * self.maze.width, 1))
    if count != 0:
        prob = 1/count
        for loc in locs:
            index = self.index(loc[0], loc[1])
            initial_distribution[index, 0] = prob

    return initial_distribution
```

Transition Matrix

There are multiple matrices necessary to set up before the algorithm starts, once of which is the transition matrix. This matrix is square, where the length (and height) is the number of tiles in the maze (includng wall tiles). This means that each diagonal component of the matrix corresponds to a certain tile (including walls) in the maze. Hence, each column contains the probability of, given that the robot is in the tile that corresponds to the diagonal that lies on that column, the robot moving to the tile that corresponds to this matrix component (not on the diagonal), as each component in each column also corresponds to a certain tile in the maze. I assume that there is a `.25` change of the robot moving in either direction, but I define the constants at the top of `HMM.py`, so they can easily be changed if you would like to mimick robot behavior that would be more inclined to move in a certain direction. Also, since the robot cannot hit any walls, if it moves into a wall, it would stay in the same place. This means that the probability of the robot staying in the same place would increase by the prbobability of the robot moving in whatever direction the wall (or end of the maze) is. Below is my code:

```
# returns transition matrix for corresponding maze
def set_transition_matrix(self):
    transition_matrix = numpy.zeros((self.state_count, self.state_count))

    # iterate through maze
    for y in range(self.maze.width):
        for x in range(self.maze.height):

            if self.maze.get_tile(x, y) is not None:
                i = self.index(x, y)

                # prob of robot bumping into wall or wall edge, and such stay in plae
                this_prob = 0

                # north
                if self.maze.get_tile(x, y - 1) is not None:
                    transition_matrix[self.index(x, y - 1), i] = PROB_NORTH
                else:
                    this_prob += PROB_NORTH

                # east
                if self.maze.get_tile(x + 1, y) is not None:
                    transition_matrix[self.index(x + 1, y), i] = PROB_EAST
                else:
                    this_prob += PROB_EAST

                # south
                if self.maze.get_tile(x, y + 1) is not None:
                    transition_matrix[self.index(x, y + 1), i] = PROB_SOUTH
                else:
                    this_prob += PROB_SOUTH

                # west
                if self.maze.get_tile(x - 1, y) is not None:
                    transition_matrix[self.index(x - 1, y), i] = PROB_WEST
                else:
                    this_prob += PROB_WEST

                transition_matrix[i, i] = this_prob

    return transition_matrix
```

Sensor Matrices

The next matrices we need to set up are the sensor matrices (one for each color). Because I wanted to allow a variable number of colors in my implemntation, I decided the best way to store these was a dictionary, where the character value of the color (which was gotten from the map) maps to the corresponding sensor matrix for that color. Sensor matrices are set up as follows. They are of the same size of the transition matrices (length = height = the number of tiles in the maze, including walls). So similar to transition matrices, each diagonal component corresponds to a certain tile. So each sensor matrix is a diagonal matrix with a color associated with it. Then each diagonal component is the probability that the robot will sense the color of this matrix if it is on top of this tile. So if the tile is the same color as the color that corresponds to the matrix, that diagonal component's value will be `0.88`. Otherwise, if the coolor is different, that diagonal component's value will be `0.04` (assuming 4 colors and those probabilities, which of course can easily be changed). Below is my code for creating this dictionary of sensor matrices:

```
# returns corresponding matrix for each color detected
def set_sensor_matrices(self):
    sensor_matrices = {}

    # iterate through each color
    for color in self.colors:
        sensor_matrix = numpy.zeros((self.state_count, self.state_count))

        # iterate through each item in maze
        for y in range(self.maze.width):
            for x in range(self.maze.height):
                c = self.maze.get_tile(x, y)
                # only change non-wall tiles
                if c != "#":
                    # set to prob success if same color, set to prob fail otherwise
                    if c == color:
                        prob = PROB_SUCCESS
                    else:
                        prob = self.prob_fail

                    # put in correct location in matrix
                    index = self.index(x, y)
                    sensor_matrix[index, index] = prob

        sensor_matrices[color] = sensor_matrix

    return sensor_matrices
```

Foward Filtering

Algorithm:

After the matrices are set up properly, the algorithm itelf is very simple. It merely starts with the initial distribution and multiplies that matrix with the transition matrix, and then multiplies the new matrix with the sensor matrix for the corresponding color. Finally, it normalizes the matrix, and has a new probability distribution for that tim step. It continues doing this for color in the color sequence, using the vector that it calculated from the previous time step. Below is the code:

```
# forward filtering algorithm
def forward(self, sequence, output_filename):
    # get initial distribution vector
    curr_matrix = self.get_initial_distribution()

    # make sure colors are all valid
    for color in sequence:
        if color not in self.maze.get_colors():
            print("Invalid color sequence.")
            return

    # iterate through sequence
    for i in range(len(sequence)):

        # the math:
        curr_matrix = numpy.dot(self.transposed_transition_matrix, curr_matrix)
        curr_matrix = numpy.dot(self.sensor_matrices[sequence[i]], curr_matrix)
        curr_matrix = curr_matrix/curr_matrix.sum(0)

    return curr_matrix
```

Forward-Backward Smoothing (Bonus)

Next as an extention, I decided to implement Forward-Backward smoothing to try to make the probability distributions more accurate. In order to do this, I had to make a couple of changes. First, I made the forward algorithm return a list of each vector at each time step, so that I could iterate through it backwards during the backwards step. Then, I also create a class-wide variable called `transition_matrix`, which is just the (non-transposed) transition matrix. I then had the proper setup to run the forward backward algorithm. Here is the corresponding code (without writing outputs to file):

```
# forward backwards smoothing algorithm
def forward_backward(self, sequence, output_filename):
    # setup, get forward list and initial backwards vector
    forward_list = self.forward(sequence, None)
    backward_vec = self.ones_vector()

    # new smoothed list, initialized to all None values
    smooth_list = []
    for i in range(len(forward_list)):
        smooth_list.append(None)

    # iterate backwards:
    for i in range(len(forward_list) - 1, 0, -1):
        smooth_list[i] = numpy.multiply(forward_list[i], backward_vec)
        smooth_list[i] = smooth_list[i]/smooth_list[i].sum(0)

        backward_vec = self.backward(backward_vec, sequence[i - 1])

    smooth_list[0] = forward_list[0]

    # write to file
    f = open(output_filename, "w")
    for i in range(len(smooth_list)):
        f.write("time step ")
        f.write(str(i))
        f.write(":\n")
        f.write(str(self.write_distribution(smooth_list[i])))
        f.write("\n\n")
    f.close()

    return smooth_list

# returns backwards vector given information
def backward(self, vec, c):
    return numpy.dot(self.transition_matrix, numpy.dot(self.sensor_matrices[c], vec))
```

Mazes

I used the following mazes and sequences to test my models:

Maze/Sequence 1:

```
rg
gr
['r', 'g', 'r', 'g']
```

This is a simple maze, that shows some weaknesses in forward filtering.

Maze/Sequence 2:

```
ggrb
ybrr
gg#r
##yb
['g', 'g', 'r', 'b', 'r', 'r', 'r', 'b', 'y']
```

This is a typical maze, with a sequence that makes sense.

Maze/Sequence 3:

```
#rrrr
yy###
bbbb#
ggg##
##r##
['r', 'g', 'g', 'b', 'r', 'y', 'y', 'y', 'r', 'b']
```

This is a maze with simple defined tiles, and a sequence that is not actually possible if all the sensor readings are correct.

Results

Foward Filtering:

Maze 1:

```
time step 0:
[[ 0.25  0.25]
 [ 0.25  0.25]]

time step 1:
[[ 0.44  0.06]
 [ 0.06  0.44]]

time step 2:
[[ 0.06  0.44]
 [ 0.44  0.06]]

time step 3:
[[ 0.44  0.06]
 [ 0.06  0.44]]

time step 4:
[[ 0.06  0.44]
 [ 0.44  0.06]]
```

Maze 2:

```
time step 0:
[[ 0.07692308  0.07692308  0.07692308  0.07692308]
 [ 0.07692308  0.07692308  0.07692308  0.07692308]
 [ 0.07692308  0.07692308  0.          0.07692308]
 [ 0.          0.          0.07692308  0.07692308]]

time step 1:
[[ 0.22680412  0.22680412  0.01030928  0.01030928]
 [ 0.01030928  0.01030928  0.01030928  0.01030928]
 [ 0.22680412  0.22680412  0.          0.01030928]
 [ 0.          0.          0.01030928  0.01030928]]

time step 2:
[[ 0.26439462  0.18152466  0.0044843   0.00071749]
 [ 0.00825112  0.00825112  0.00071749  0.00071749]
 [ 0.26439462  0.26439462  0.          0.00071749]
 [ 0.          0.          0.00071749  0.00071749]]

time step 3:
[[  8.60099622e-02   5.48995191e-02   4.93601855e-01    7.94400550e-04]
 [  6.52696668e-02   5.44486431e-02   3.73153555e-02    7.55754036e-03]
 [  9.59292339e-02   9.59292339e-02   0.00000000e+00    7.55754036e-03]
 [  0.00000000e+00   0.00000000e+00   3.43524562e-04   3.43524562e-04]]

time step 4:
[[  1.45659385e-02   3.43453890e-02   2.92431898e-02   5.51375731e-01]
 [  1.50379481e-02   2.77924827e-01   2.95578629e-02   2.65331482e-03]
 [  1.76002860e-02   1.70608464e-02   0.00000000e+00   1.14737938e-03]
 [  0.00000000e+00   0.00000000e+00   6.85002617e-05   9.41878598e-03]]

time step 5:
[[  2.10818416e-03   9.56096017e-03   3.80728397e-01   3.04660301e-02]
 [  8.72992344e-03   2.57771689e-03   2.00476108e-01   3.45410845e-01]
 [  1.80703142e-03   8.85122945e-03   0.00000000e+00   8.48670790e-03]
 [  0.00000000e+00   0.00000000e+00   2.58418310e-04   5.38448115e-04]]

time step 6:
[[  3.98627065e-04   6.99542656e-03   2.42058566e-01   1.39398592e-02]
 [  2.69612763e-04   4.03135770e-03   3.62053668e-01   2.27878751e-01]
 [  3.75389526e-04   3.91187632e-04   0.00000000e+00   1.41410330e-01]
 [  0.00000000e+00   0.00000000e+00   2.32670604e-05   1.73958331e-04]]

time step 7:
[[  1.32035103e-04   4.15127323e-03   2.25199034e-01   8.15268307e-03]
 [  8.31123850e-05   6.05468955e-03   3.01211376e-01   2.68518661e-01]
 [  2.31172480e-05   8.49815658e-05   0.00000000e+00   1.84063108e-01]
 [  0.00000000e+00   0.00000000e+00   3.99201696e-06   2.32193690e-03]]

time step 8:
[[  1.79295921e-04   9.38785011e-03   2.14716543e-02   4.47218173e-01]
 [  2.50819632e-04   2.67907300e-01   3.19249819e-02   3.03690387e-02]
 [  8.54253498e-06   2.49018709e-04   0.00000000e+00   2.54674376e-02]
 [  0.00000000e+00   0.00000000e+00   9.30232702e-05   1.65472864e-01]]

time step 8:
[[  7.62215546e-04   2.27923780e-02   3.88838436e-02   7.21464638e-02]
 [  4.50105671e-01   3.18789972e-03   2.68124032e-02   4.07881487e-02]
 [  3.94114985e-05   2.04645272e-02   0.00000000e+00   1.88148620e-02]
 [  0.00000000e+00   0.00000000e+00   2.78021276e-01   2.71808993e-02]]
```

Maze 3:

```
time step 0:
[[ 0.          0.07142857  0.07142857  0.07142857  0.07142857]
 [ 0.07142857  0.07142857  0.          0.          0.        ]
 [ 0.07142857  0.07142857  0.07142857  0.07142857  0.        ]
 [ 0.07142857  0.07142857  0.07142857  0.          0.        ]
 [ 0.          0.          0.07142857  0.          0.        ]]

time step 1:
[[ 0.          0.18487395  0.18487395  0.18487395  0.18487395]
 [ 0.00840336  0.00840336  0.          0.          0.        ]
 [ 0.00840336  0.00840336  0.00840336  0.00840336  0.        ]
 [ 0.00840336  0.00840336  0.00840336  0.          0.        ]
 [ 0.          0.          0.18487395  0.          0.        ]]

time step 2:
[[ 0.          0.05731394  0.07527802  0.07527802  0.07527802]
 [ 0.00342173  0.0213858   0.          0.          0.        ]
 [ 0.00342173  0.00342173  0.00342173  0.00342173  0.        ]
 [ 0.07527802  0.07527802  0.4704876   0.          0.        ]
 [ 0.          0.          0.05731394  0.          0.        ]]

time step 3:
[[  0.00000000e+00   6.09530390e-03   8.16820077e-03   8.68642499e-03   8.68642499e-0
3]
 [  9.13061718e-04   2.46773437e-03   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  2.46773437e-03   2.98595859e-03   1.38686672e-02   3.94837500e-04   0.00000000e+0
0]
 [  1.45497619e-01   3.96318140e-01   3.84917208e-01   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   1.85326851e-02   0.00000000e+00   0.00000000e+0
0]]

time step 4:
[[  0.00000000e+00   9.25337748e-04   1.26146043e-03   1.38750644e-03   1.40851411e-0
3]
 [  2.74100047e-04   5.05184392e-04   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  1.35437435e-01   3.70219130e-01   3.58664913e-01   1.34249001e-02   0.00000000e+0
0]
 [  2.79622062e-02   3.76887564e-02   3.29830388e-02   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   1.78575180e-02   0.00000000e+00   0.00000000e+0
0]]

time step 5:
[[ 0.          0.01277917  0.01708366  0.01923591  0.01982963]
 [ 0.02191778  0.05972375  0.          0.          0.        ]
 [ 0.08573286  0.08547648  0.12449688  0.06406198  0.        ]
 [ 0.03678109  0.07528873  0.07181073  0.          0.        ]
 [ 0.          0.          0.30578134  0.          0.        ]]

time step 6:
[[ 0.          0.00870977  0.00563112  0.00641413  0.00669829]
 [ 0.35432963  0.33674347  0.          0.          0.        ]
 [ 0.0195617   0.02937488  0.02942625  0.0269449   0.        ]
 [ 0.01995952  0.0229182   0.04912608  0.          0.        ]
 [ 0.          0.          0.08416206  0.          0.        ]]

time step 7:
[[ 0.          0.00863301  0.00063312  0.00060364  0.00063606]
 [ 0.56216766  0.38490385  0.          0.          0.        ]
 [ 0.010155    0.00980526  0.00323616  0.00264563  0.        ]
 [ 0.0019771   0.0029124   0.00445412  0.          0.        ]
 [ 0.          0.          0.00723698  0.          0.        ]]

time step 8:
[[  0.00000000e+00   7.16948370e-03   1.86940641e-04   4.40785211e-05   4.47081019e-0
5]
 [  5.94960788e-01   3.78072042e-01   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  1.03964760e-02   7.14108416e-03   3.58492570e-04   1.98869211e-04   0.00000000e+0
0]
 [  3.02967372e-04   3.40830721e-04   3.17527776e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   4.65711299e-04   0.00000000e+00   0.00000000e+0
0]]

time step 9:
[[  0.00000000e+00   6.93727359e-01   1.34071437e-02   5.65102346e-04   3.14887473e-0
4]
 [  1.26774662e-01   7.93024017e-02   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  4.92195686e-02   3.12575590e-02   6.43834728e-04   7.67126615e-05   0.00000000e+0
0]
 [  9.11077433e-04   6.50777203e-04   1.19077878e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   3.02983679e-03   0.00000000e+00   0.00000000e+0
0]]

time step 10:
[[  0.00000000e+00   1.25544440e-01   6.11627675e-02   1.25973561e-03  1.28055092e-04
]
 [  3.24064887e-02   7.89707316e-02   0.00000000e+00   0.00000000e+00  0.00000000e+00
]
```

```
 [  3.88430666e-01   2.42236967e-01   5.98931540e-02   1.63082786e-03  0.00000000e+00
]
 [  4.38444988e-03   2.79377401e-03   3.76890641e-04   0.00000000e+00  0.00000000e+00
]
 [  0.00000000e+00   0.00000000e+00   7.81053212e-04   0.00000000e+00  0.00000000e+00
]]
```

Forward-Backward Smoothing:

Maze 1:

```
time step 0:
[[ 0.25  0.25]
 [ 0.25  0.25]]

time step 1:
[[ 0.44  0.06]
 [ 0.06  0.44]]

time step 2:
[[ 0.06  0.44]
 [ 0.44  0.06]]

time step 3:
[[ 0.44  0.06]
 [ 0.06  0.44]]

time step 4:
[[ 0.06  0.44]
 [ 0.44  0.06]]
```

Maze 2:

```
time step 0:
[[ 0.07692308  0.07692308  0.07692308  0.07692308]
 [ 0.07692308  0.07692308  0.07692308  0.07692308]
 [ 0.07692308  0.07692308  0.          0.07692308]
 [ 0.          0.          0.07692308  0.07692308]]

time step 1:
[[ 0.39500254  0.41785307  0.02136826  0.0066857 ]
 [ 0.00205706  0.01920837  0.00593074  0.00802985]
 [ 0.04927902  0.06191566  0.          0.00743728]
 [ 0.          0.          0.00118747  0.00404497]]
```

```
time step 2:
[[  5.19563621e-02   7.53682289e-01   3.23370876e-02   6.64707870e-03]
 [  8.89041347e-04   2.48387247e-02   8.11442574e-03   8.27472042e-03]
 [  4.83936782e-02   5.19947571e-02   0.00000000e+00   9.08318799e-03]
 [  0.00000000e+00   0.00000000e+00   4.80617734e-04   3.30802924e-03]]

time step 3:
[[  1.78935746e-03   4.59314505e-02   7.56444754e-01   2.30976467e-03]
 [  4.98322504e-02   6.25142726e-03   3.86486827e-02   1.30087512e-02]
 [  1.03009590e-04   7.32449717e-02   0.00000000e+00   1.12637320e-02]
 [  0.00000000e+00   0.00000000e+00   3.75311381e-04   7.96536795e-04]]

time step 4:
[[  1.33170823e-04   1.33770875e-02   3.18393218e-02   6.85937761e-01]
 [  1.43644637e-04   1.93227229e-01   5.59899869e-02   6.62794439e-03]
 [  1.89357580e-05   1.67817739e-04   0.00000000e+00   3.17531354e-03]
 [  0.00000000e+00   0.00000000e+00   1.18296501e-06   9.36060393e-03]]

time step 5:
[[  1.19871075e-05   2.99755132e-03   2.39794639e-01   2.53300604e-02]
 [  1.36249286e-04   8.51154631e-04   2.32808933e-01   4.83779458e-01]
 [  7.69929855e-06   1.41091524e-04   0.00000000e+00   1.38256362e-02]
 [  0.00000000e+00   0.00000000e+00   5.86036951e-06   3.09680485e-04]]

time step 6:
[[  2.70989200e-05   1.00924573e-03   1.84682371e-01   3.30772302e-03]
 [  1.10024347e-05   2.83985202e-03   3.02721025e-01   3.07467975e-01]
 [  2.55192174e-05   2.49578771e-05   0.00000000e+00   1.97745450e-01]
 [  0.00000000e+00   0.00000000e+00   3.96469492e-06   1.33815044e-04]]

time step 7:
[[  3.16971485e-05   7.35450707e-03   6.84337310e-02   4.55850125e-03]
 [  1.57851771e-04   6.80765126e-04   5.14412345e-01   8.15977472e-02]
 [  5.54966693e-06   1.50555623e-04   0.00000000e+00   3.14345148e-01]
 [  0.00000000e+00   0.00000000e+00   9.11036989e-06   8.26249109e-03]]

time step 8:
[[  3.41748931e-04   2.86301013e-03   6.54820466e-03   1.36388006e-01]
 [  4.78077474e-04   5.10647608e-01   9.73615319e-03   9.26163758e-03]
 [  1.62825913e-05   7.59431690e-05   0.00000000e+00   7.76679763e-03]
 [  0.00000000e+00   0.00000000e+00   4.75185470e-04   3.15401344e-01]]

time step 9:
[[  7.62215546e-04   2.27923780e-02   3.88838436e-02   7.21464638e-02]
```

```
 [  4.50105671e-01   3.18789972e-03   2.68124032e-02   4.07881487e-02]
 [  3.94114985e-05   2.04645272e-02   0.00000000e+00   1.88148620e-02]
 [  0.00000000e+00   0.00000000e+00   2.78021276e-01   2.71808993e-02]]
```

Maze 3:

```
time step 0:
[[ 0.          0.07142857  0.07142857  0.07142857  0.07142857]
 [ 0.07142857  0.07142857  0.          0.          0.        ]
 [ 0.07142857  0.07142857  0.07142857  0.07142857  0.        ]
 [ 0.07142857  0.07142857  0.07142857  0.          0.        ]
 [ 0.          0.          0.07142857  0.          0.        ]]

time step 1:
[[ 0.          0.01660409  0.01075575  0.00510371  0.00174282]
 [ 0.00212794  0.00159142  0.          0.          0.        ]
 [ 0.06823744  0.04679518  0.02118201  0.00048057  0.        ]
 [ 0.17777781  0.13152121  0.0653567   0.          0.        ]
 [ 0.          0.          0.45072336  0.          0.        ]]

time step 2:
[[  0.00000000e+00   1.29546756e-02   1.08237519e-02   4.36039311e-03   8.70298235e-0
4]
 [  1.11571803e-03   6.12568486e-03   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  5.30022587e-03   3.49642536e-03   1.16411834e-03   2.52715126e-04   0.00000000e+0
0]
 [  2.63037375e-01   1.79736260e-01   4.98842412e-01   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   1.19199464e-02   0.00000000e+00   0.00000000e+0
0]]

time step 3:
[[  0.00000000e+00   1.24043523e-02   1.06452267e-02   2.97634271e-03   1.68283410e-0
4]
 [  2.48284902e-03   6.07524612e-03   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  8.39689910e-03   8.10105354e-03   2.15571853e-02   1.32377603e-04   0.00000000e+0
0]
 [  2.96684735e-01   5.04449643e-01   1.25810028e-01   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   1.15779073e-04   0.00000000e+00   0.00000000e+0
0]]

time step 4:
```

```
[[  0.00000000e+00   1.49045208e-02   9.97101016e-03   3.23671892e-04   1.14853721e-0
4]
 [  1.11634207e-03   5.09385140e-03   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  3.00738046e-01   5.12707480e-01   1.24503470e-01   1.28902038e-04   0.00000000e+0
0]
 [  1.53078823e-02   1.33992917e-02   9.43635519e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   7.47042780e-04   0.00000000e+00   0.00000000e+0
0]]

time step 5:
[[  0.00000000e+00   2.72922803e-02   8.14454548e-04   1.23074242e-04   1.10276300e-0
4]
 [  1.84397271e-01   3.15730947e-01   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  2.78621440e-01   1.80109921e-01   5.47775106e-03   3.37278052e-04   0.00000000e+0
0]
 [  2.12465260e-03   3.29580699e-03   3.82553977e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   1.18229351e-03   0.00000000e+00   0.00000000e+0
0]]

time step 6:
[[  0.00000000e+00   4.27913006e-03   1.48695625e-04   1.03925932e-04   1.09757789e-0
4]
 [  5.97167236e-01   3.68759112e-01   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  1.23751654e-02   1.38612066e-02   6.43987239e-04   3.59203589e-04   0.00000000e+0
0]
 [  2.73398115e-04   4.32947084e-04   4.92820031e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   9.93414174e-04   0.00000000e+00   0.00000000e+0
0]]

time step 7:
[[  0.00000000e+00   5.21096238e-03   1.03254064e-04   1.03058060e-04   1.08593743e-0
4]
 [  6.48529736e-01   3.34262346e-01   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  3.80240449e-03   5.39422234e-03   3.62971779e-04   3.91051939e-04   0.00000000e+0
0]
 [  1.21065934e-04   2.15417604e-04   4.54212624e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   9.40703507e-04   0.00000000e+00   0.00000000e+0
0]]

time step 8:
[[  0.00000000e+00   1.41153420e-02   4.48282797e-04   1.05700091e-04   1.07209823e-0
4]
 [  4.90432673e-01   4.73912395e-01   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  1.00572716e-02   7.92971243e-03   5.51943064e-04   4.48436780e-04   0.00000000e+0
0]
 [  2.49739313e-04   2.80950485e-04   4.43446533e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   9.16896468e-04   0.00000000e+00   0.00000000e+0
0]]

time step 9:
[[  0.00000000e+00   2.35362019e-01   4.54866365e-03   1.91723200e-04   1.06832390e-0
4]
 [  2.68819047e-01   1.68156600e-01   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  1.92036244e-01   1.21955238e-01   3.65878398e-03   5.72581472e-04   0.00000000e+0
0]
 [  1.93189210e-03   1.37993906e-03   2.52498418e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   1.02793770e-03   0.00000000e+00   0.00000000e+0
0]]

time step 10:
[[  0.00000000e+00   1.25544440e-01   6.11627675e-02   1.25973561e-03   1.28055092e-0
4]
 [  3.24064887e-02   7.89707316e-02   0.00000000e+00   0.00000000e+00   0.00000000e+0
0]
 [  3.88430666e-01   2.42236967e-01   5.98931540e-02   1.63082786e-03   0.00000000e+0
0]
 [  4.38444988e-03   2.79377401e-03   3.76890641e-04   0.00000000e+00   0.00000000e+0
0]
 [  0.00000000e+00   0.00000000e+00   7.81053212e-04   0.00000000e+00   0.00000000e+0
0]]
```

Conclusions

Forward Filtering

The forward filtering method provides some interesting insights, as well as certain limitations. The first success is that, intuitively, the probability distributions make sense, in that it is more likely that the robot is on tiles of the color that it senses. Another interesting point is `time step 5` of `Maze 3`. At this point, the robot has sensed (at some point in the sequence) at least one color incorrectly. The distribution is quite divided, which makes sense, since it is much more difficult to determine where the robot actually is at this point in time. It is still most likely that it ends up on a `r` tile of course, which is reflected in the probability distribution. Interestingly, it is not very likely that the robot is on a `r` tile on the top row. This makes sense, because at this point in time there has been no `y` tile in the sensed sequence, which would be necessary in order to reach the `r` tiles in the top row. It is encouraging that this metod was able to pick up on that, and determine that is was unlikely that the robot would be in those locations, despite the fact that they shared the same color as the one that the robot sensed.

Forward-Backward Smoothing

This will attempt to assess the improvements that the Forward-Backward Smoothing makes in creating probability distributions over the Forward Filtering algorithm. Note that as I do not have the mathematical ability to calculate the true probability distribution for each time step at each maze, so it is rather difficult to know whether or not the changes the the Forward-Backward Smoothing makes are actually an improvement or not. But here are the changes that it caused. Maze 1 was completely the same as the Foward Filtering algorithm. I think that it is clear that the probabilies show approach `.25` for each tile, as the number of switches increase (beceause the robot becomes more and more likely to provide an innacurate sensor reading, so you become less and less certain of where the robot actually is), so it would be interesting to find an algorithm that takes that into account in the future. Next, in Maze 2, it concluded that it was far more likely that the robot was in the top right corner green tiles during time step 1, rather than the other green times. The forward propagation thought that these were all of equal probability. This is because at that point, all could be considered to have equal probability, but after propagating forwards through the entire sequence and then backwards, you can see that in hindsight, its actually quite unlikely that it is in the more bottom-ish green tiles. Draw out the sequence on the map if you would like to see it. This was an encouraging imporovement in the Forward-Backward Smoothing algorithm over the Foward Filtering algorithm. Maze 3 tells a similar story to Maze 2. Foward-Backward Smoothing decides that it is far and away most likely that at time step 1, the robot is at the bottom of the map, and in fact more likely that it is on a green tile rather than the red tiles on the top of the map. This shows that it actually is taking into account some of the error that the robot makes in its sensing now, which is a very encouraging sign and improvement.

Literature Review (Bonus)

This was the second time I have used Hidden Markov Models, the first time was for a Part of Speach Identifier in CS 10. As that is a much different application than this, I was interested in what other potential applications Hidden Markov Models were used for. As such, I decided to read "A hidden Markov model for prediction transmembrane helices in protein sequences" by Sonnhammer, Heijne and Krogh. This article provided a novel method for modeling and predicting the location and orientation of alpha helices in proteins that go through the cell membrane. First, the article outlines previous methods in which this task has been accomplished. Apparently the weakness with traditioal methods is that they use fixed hydrophibicity thresholds, or how hydrophobic a certain region of the protein can be. This is important because the cell membrane is a non-polar environment.

In order to make their HMM model better, they tried to map the architecture of the biological protein more closely. This means that this model has more states, and thus can adapt better and become closer to mimicking the real life protein. Hence, the prediction should be more accurate. They also programmed each state with all of the different amino acid probabilities of occuring, depending on what part of the protein the state was located. For example, some parts are the hydrophillic caps on each side of the protein, which have different amino acids that are more likely to be present, while the hydrophobic middle is more likely to have other amino acids present in greater numbers. This is similar to how my robot had a likelihood to provide the correct (and incorrect) color from its sensor. In all, their model achieved about 77% accuracy, which was a significant improvement over traditional methods, which only achieved about 56% accuracy most of the time. This seems like a huge improvement from their application of HMMs.

I have taken bio classes in the past (including Cell Bio here at Dartmouth), so I found that this article was a interesting way to combine two of my interests (CS and Biology). I find it very fascinating that HMMs can be used to predict how proteins can fold, as I am aware of just how complicated such a problem can be. It makes sense that computers and concepts such as HMMs would be used to provide more accurate potential solutions.

Running Tests

In order to run the tests, just run the `HMM.py` file.