HMM

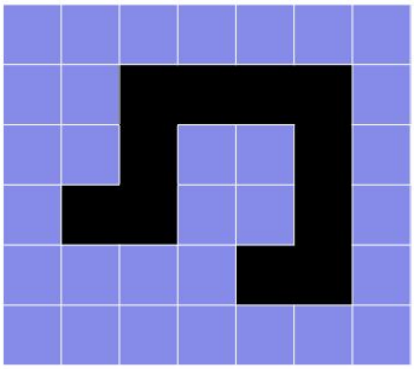


Figure 1: Windy Maze

We assume that a robot aims to locate itself in the windy maze defined in P1. The maze map is shown in Figure 1 for your convenience. The robot will perform two kinds of actions: `sensing` and `moving`.

Sensing In a square, the robot will sense the four directions to see if there is an obstacle in this direction. We assume that the whole maze is surrounded by obstacles and the black squares are also obstacle. However, the sensing is not perfect. We assume that the robot can detect the obstacle with 85% if there is and might mistake an open square as a obstacle with 10%. The detections in all directions are done independently.

Moving In the windy situation, the robot can drift to the left or the right with probability 0.1. If the drifting direction is an obstacle, it will be bounced back to the original position. For example, in the square of left bottom, if the robot moves northward, it will reach the square to the north with 80% and reach the square to the east with 10% and be bounced back to the original position with 10%.

We assume that the robot initially stays in one open square, but it doesn't know its exact location except that it knows that it can't be in any obstacle square. Then the robot performs the following sequence of actions:

1. Sensing: [-, -, -, O]
2. Moving northward
3. Sensing: [-, O, -, -]
4. Moving eastward
5. Sensing: [-, O, -, -]
6. Moving eastward
7. Sensing: [-, -, O, -]

where (W,N,E,S) indicates the observation at Directions (Westward, Northward, Eastward, Southward), respectively. "-" indicates no obstacle is observed and "O" indicates an obstacle is observed.

You are expected to report all the prior and posterior probabilities of the latest robot location at each square after each action as follows (3.23 means 3.23%):

Evidence Conditional Probability

- For every valid grid space (non-obstacles), the true/accurate flags are first set
- Then comparing what was the evidence from the sensor (which may have errors), we determine whether it was correctly sensed and use the appropriate probabilities for that grid
    - For example, if a north obstacle is there, but sensed as an open grid space, it would be a 15% probability ("o" marked as "-"). If north was an open space but sensed as an obstacle, it would be a 10% probability ("-" marked as "o")
- For each space, the evidence conditional probability is stored into its respective slot in evGrid
- While the calculation for each grid space is performed, evNormTerm (the term used to normalize the probability) is summed for use in filtering later

```
70  void evidenceProb(std::string w, std::string n, std::string e, std::string s)
71  {
72      //Calculate P(Z=(w,n,e,s)|S)
73      float correctSpace = 0.9;       //"-" sensed as "-" = 90 %
74      float incorrectSpace = 0.1;     //"-" sensed as "O" = 10 %
75      float correctObstacle = 0.85;   //"O" sensed as "O" = 85 %
76      float incorrectObstacle = 0.15; //"O" sensed as "-" = 15 %
77      bool westObstacle = false, northObstacle = false, eastObstacle = false, southObstacle = false;
78      float west = 0.0, north = 0.0, east = 0.0, south = 0.0, evidenceProbability = 0.0;
79
80      evNormTerm = 0.0;
81      for (int i = 0; i < row; i++)
82      {
83          for (int j = 0; j < col; j++)
84          {
85              //For each grid space, reset flags
86              westObstacle = false;
87              northObstacle = false;
88              eastObstacle = false;
89              southObstacle = false;
90
91              if (grid[i][j] != "####")
92              {
93                  //The accurate obstacle flags are set here
94                  if (j - 1 < 0 || grid[i][j - 1] == "####")
95                  {
96                      westObstacle = true;
97                  }
98                  if (i - 1 < 0 || grid[i - 1][j] == "####")
99                  {
100                     northObstacle = true;
101                 }
102                 if (j + 1 >= col || grid[i][j + 1] == "####")
103                 {
104                     eastObstacle = true;
105                 }
106                 if (i + 1 >= row || grid[i + 1][j] == "####")
107                 {
108                     southObstacle = true;
109                 }
```

Filtering

- Uses the evidence conditional probability, multiplied by the prior, and normalized with evNormTerm
    - Essentially calculate P(Z|S)P(S) / normTerm for every valid grid space
- Posterior calculation updates as the new prior for future calculations

```
351  void filter(std::string w, std::string n, std::string e, std::string s)
352  {
353      //Calculate P(Z|S)P(S)
354      //Evidence probability * prior / normalization term = individual position %
355
356      evidenceProb(w, n, e, s);
357      //For each grid space, calculate its normalized probability
358      for (int i = 0; i < row; i++)
359      {
360          for (int j = 0; j < col; j++)
361          {
362              if (grid[i][j] != "####")
363              {
364                  //Grid displays %, so *100 for value
365                  grid[i][j] = std::to_string(evGrid[i][j]*priorGrid[i][j]/evNormTerm*100);
366              }
367          }
368      }
369      updatePrior();
370  }
```

Transitional Probability

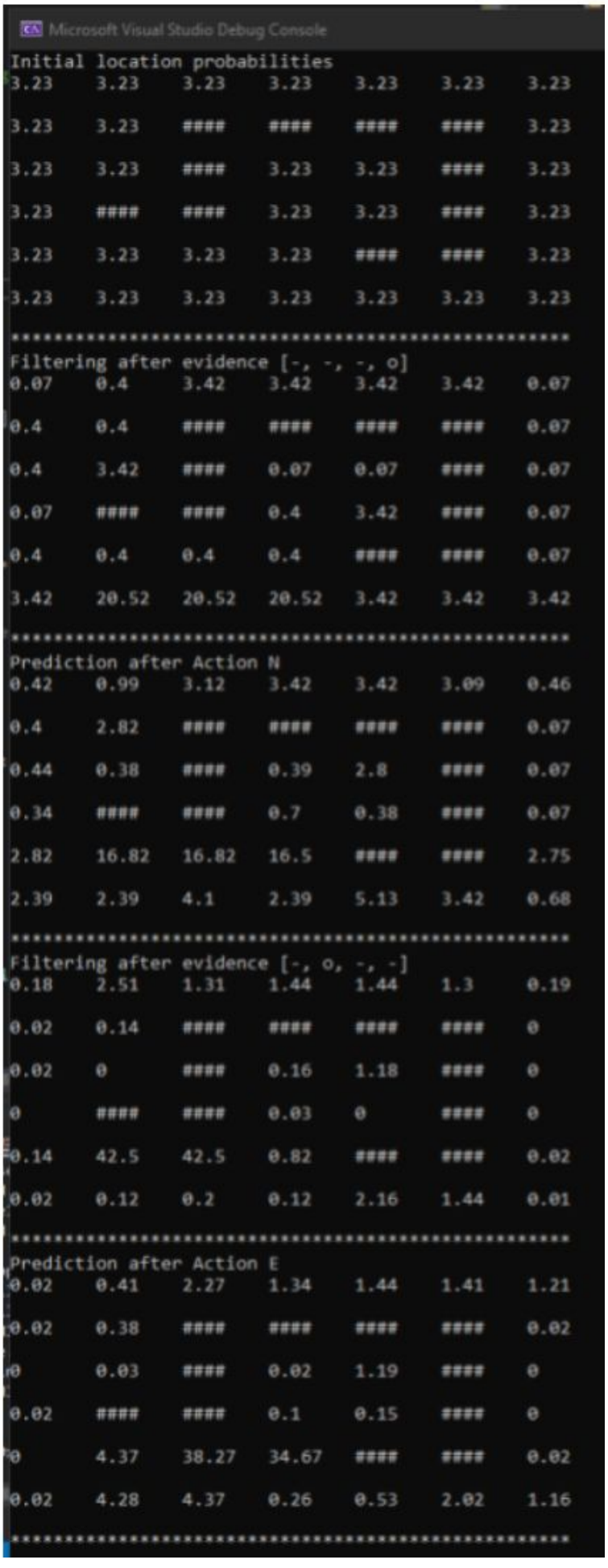
- Take into account how the desired movement direction would "enter" a specific grid space
    - For example, if the movement is north:
        - The potential movement entering the specific grid space comes from:
            - Drift with 10% probability from grid spaces to the left and to the right of the specific grid space
                - Or 10% probability from itself if drift causes it bounce into wall/obstacle
            - 80% from itself (if bouncing into a wall/obstacle)
            - 80% from the grid space below it (if it's a valid grid space)
        - Situations where it doesn't apply would be a 0% probability, and thus not part of the total probability summation

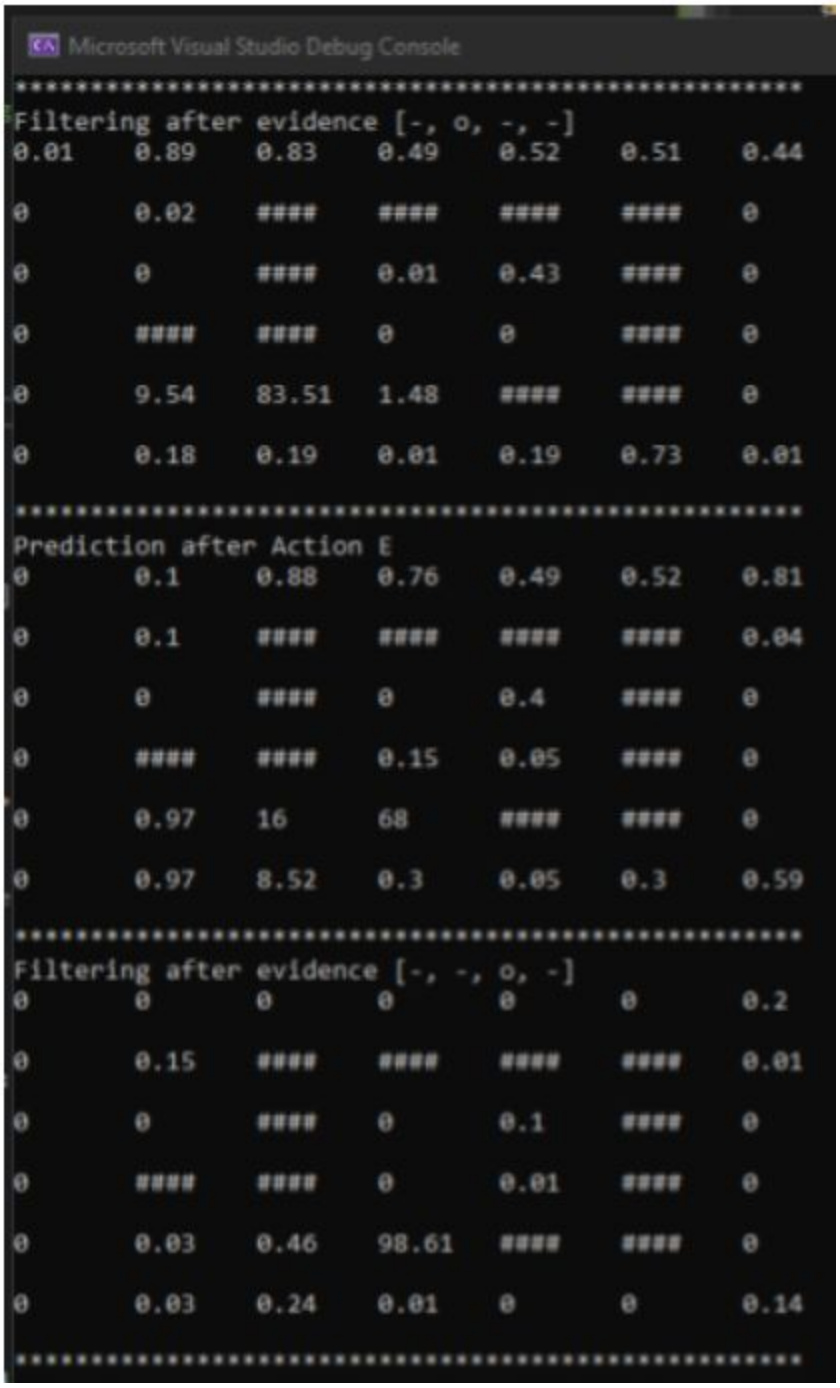
```
float transition(std::string dir, int i, int j)
{
    /*
    Given desired direction and specific grid space, calculate the total prob
    Moving towards desired direction - 80%
    Drifting left or right - 10% each
    */
    float driftProb = 0.1;
    float desiredProb = 0.8;
    float totalProb = 0.0;

    //Given desired direction, account for whichever other spaces lead into each grid space
    if (dir == "w")
    {
        /*Desired direction is West
        80% movement from East - desiredProb
        80% movement from bouncing back - desiredProb
        10% movement from North - driftProb
        10% movement from South - driftProb
        */

        //If movement coming from east grid is possible, take that into account. Otherwise its 0%
        if (j + 1 < col && grid[i][j + 1] != "####")
        {
            totalProb += desiredProb * priorGrid[i][j + 1];
        }
        //If moving west bounces into the wall or an obstacle
        if (j - 1 < 0 || grid[i][j - 1] == "####")
        {
            totalProb += desiredProb * priorGrid[i][j];
        }
        //If drifting north bounces into the wall or an obstacle
        if (i - 1 < 0 || grid[i - 1][j] == "####")
        {
            totalProb += driftProb * priorGrid[i][j];
        }
        else
        {
            totalProb += driftProb * priorGrid[i-1][j];
        }
        //If drifting south bounces into the wall or an obstacle
        if (i + 1 >= row || grid[i + 1][j] == "####")
        {
            totalProb += driftProb * priorGrid[i][j];
        }
        else
        {
            totalProb += driftProb * priorGrid[i+1][j];
        }
    }
```

Prediction

- Uses the transitional probability model to go through every valid grid space
    - Feed in the desired movement direction and check for each grid space the potential movement from each direction (drift and desired movement)
- Posterior calculation updates as the new prior for future calculations

```
372  void predictAfterAction(std::string dir)
373  {
374      for (int i = 0; i < row; i++)
375      {
376          for (int j = 0; j < col; j++)
377          {
378              if (grid[i][j] != "####")
379              {
380                  grid[i][j] = std::to_string(transition(dir, i, j));
381              }
382          }
383      }
384
385      updatePrior();
386  }
```


Microsoft Visual Studio Debug Console
Initial location probabilities
3.23 3.23 3.23 3.23 3.23 3.23 3.23
3.23 3.23 #### #### #### #### 3.23
3.23 3.23 #### 3.23 3.23 #### 3.23
3.23 #### #### 3.23 3.23 #### 3.23
3.23 3.23 3.23 3.23 #### #### 3.23
3.23 3.23 3.23 3.23 3.23 3.23 3.23
****
Filtering after evidence [-, -, -, o]
0.07 0.4 3.42 3.42 3.42 3.42 0.07
0.4 0.4 #### #### #### #### 0.07
0.4 3.42 #### 0.07 0.07 #### 0.07
0.07 #### #### 0.4 3.42 #### 0.07
0.4 0.4 0.4 0.4 #### #### 0.07
3.42 20.52 20.52 20.52 3.42 3.42 3.42
****
Prediction after Action N
0.42 0.99 3.12 3.42 3.42 3.09 0.46
0.4 2.82 #### #### #### #### 0.07
0.44 0.38 #### 0.39 2.8 #### 0.07
0.34 #### #### 0.7 0.38 #### 0.07
2.82 16.82 16.82 16.5 #### #### 2.75
2.39 2.39 4.1 2.39 5.13 3.42 0.68
****
Filtering after evidence [-, o, -, -]
0.18 2.51 1.31 1.44 1.44 1.3 0.19
0.02 0.14 #### #### #### #### 0
0.02 0 #### 0.16 1.18 #### 0
0 #### #### 0.03 0 #### 0
0.14 42.5 42.5 0.82 #### #### 0.02
0.02 0.12 0.2 0.12 2.16 1.44 0.01
****
Prediction after Action E
0.02 0.41 2.27 1.34 1.44 1.41 1.21
0.02 0.38 #### #### #### #### 0.02
0 0.03 #### 0.02 1.19 #### 0
0.02 #### #### 0.1 0.15 #### 0
0 4.37 38.27 34.67 #### #### 0.02
0.02 4.28 4.37 0.26 0.53 2.02 1.16
****


Microsoft Visual Studio Debug Console
****
Filtering after evidence [-, o, -, -]
0.01 0.89 0.83 0.49 0.52 0.51 0.44
0 0.02 #### #### #### #### 0
0 0 #### 0.01 0.43 #### 0
0 #### #### 0 0 #### 0
0 9.54 83.51 1.48 #### #### 0
0 0.18 0.19 0.01 0.19 0.73 0.01
****
Prediction after Action E
0 0.1 0.88 0.76 0.49 0.52 0.81
0 0.1 #### #### #### #### 0.04
0 0 #### 0 0.4 #### 0
0 #### #### 0.15 0.05 #### 0
0 0.97 16 68 #### #### 0
0 0.97 8.52 0.3 0.05 0.3 0.59
****
Filtering after evidence [-, -, o, -]
0 0 0 0 0 0 0.2
0 0.15 #### #### #### #### 0.01
0 0 #### 0 0.1 #### 0
0 #### #### 0 0.01 #### 0
0 0.03 0.46 98.61 #### #### 0
0 0.03 0.24 0.01 0 0 0.14
****