Probabilistic reasoning over time Solution

Boying Shi

March 7, 2014

1 Introduction

In this homework, we are asked to design and implement a blind robot localization problem by applying knowledge of Hidden Markov Model for our basic approach. As the assignment noted, we need to not only print of a sequence of probability distributions describing the possible locations of the robot at each time step given the information of color of each cell of the maze as well as the information of the wall.

I finished designing and implementing a project of robot localization using reasoning over time and filtering algorithm. To implement my program, I used hidden markov models as my basic approach and finish it by forming transition matrix(transition model) and sensor model. I printed out each step of sensor model and the entire maze with value of color for each cell as well as the current probability for each cell. Also, I printed out the possible localization of the blind robot at each time step.

Not using some fancy for presenting the maze and probability distribution, I simply using ASCII art to present necessary information of the maze.

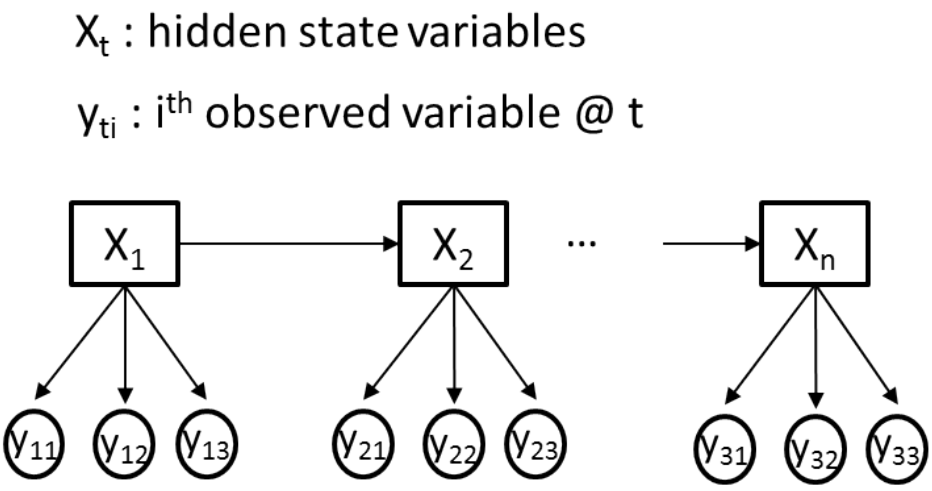

Figure 1: Probabilistic reasoning over time[1]

2 Program Design

To implement this problem, I should firstly design it by myself so I will discuss my idea of designing in this section. Clearly, according to the basic approach of hidden markov model, I will have a Sensor Model class and a Transition Model class to help me calculate probability at each time step. In this two classes, it includes functions like updating each time step's probability at each cell and setting transition matrix and setting relaitionships(neighbors). Also, it includes other helpers like calculateTransitionMatrix, getRelationship and setRelationship.

Except these two main functional classes, I also need a class named Maze to help me consider about attributes like height, weight, colors and probability at each cell. Also, it need to care about functions like initial color and position for each cell, print maze information and so on.

Then, to better finish my work, I also need a class named Position, to help me represent position of each location like the (x, y) style. Also, it can help me take care of get and set X value as well as get and set Y value. Finally, I added a class named Driver to do the following works: initial maze information and run main function.

To better explain my design idea, I will show my class diagram of this program below:

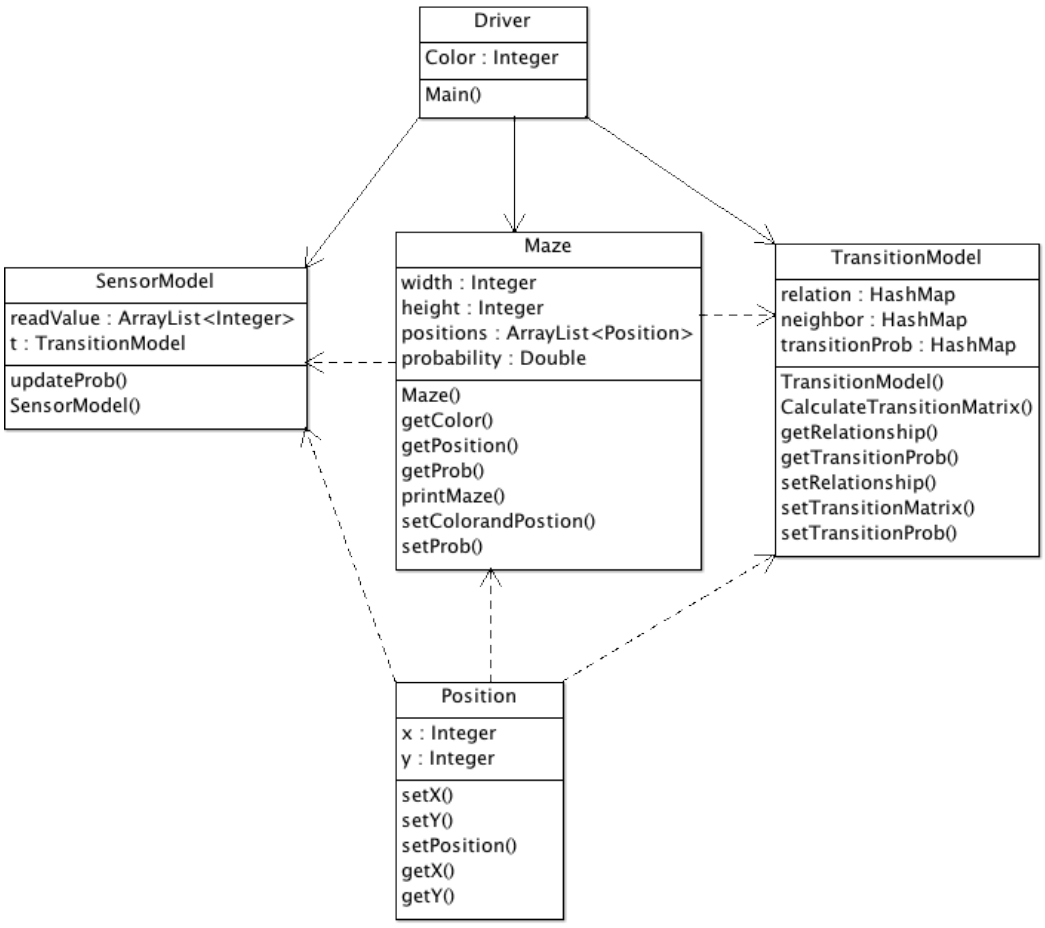

Figure 2: Program Design

3 Implemenation

To better describe my works of implementation, I will introduce my work by each class. And the general idea of this is to read a sequence of data for the sensor, and the sensor will update each probability at each cell for each time step. At each time step, each color on the maze that is the "right" color recording to the sensor will get a new probability by multiple its current probability with 0.88, while the other "wrong" color will get a new probability by multiple its current probability with 0.04 (0.12/3). Also, if the sequence of input colors is bigger than 1, it means the robot will take at least one movement. So, it need to multiple its transition matrix value. At first, the probability distribution of the robot will be all 0.0625 at each cell. Then, if at the next time step, the transition matrix will update according each cell and its neighbors possible movements.

3.1 Maze.java

Maze class is used to deal with information about the maze and do some based calculations. My code is as below:

Listing 1: Maze.java

```
1  public class Maze {
2    public int width;
3    public int height;
4    public int[][] color;
5    public double[][] prob;
6    public ArrayList<Position> p;
7
8    //constructor of maze, to initial maze information
9    public Maze(int x, int y){
10     this.width = x;
11     this.height = y;
12     this.color = new int[x][y];
13     this.prob = new double[x][y];
14     this.p = new ArrayList<Position>();
15     for(int i = 0;i<x;i++){
16       for(int j=0;j<y;j++){
17         this.color[i][j] = new Random().nextInt(4);
18         this.prob[i][j] = 1.0/(this.width*this.height);
19           }
20         }
21      }
22
23    //set color and position for each cell
24    public void setColorandPosition(int posx, int posy, int color){
25      this.color[posx][posy] = color;
26      this.p.add(new Position(posx,posy));
27      }
28
29    //get color for current position
30    public int getColor(int posx, int posy){
31      return this.color[posx][posy];
32     }
33
34   //set probability of current position
```

```
35    public void setProb(int posx, int posy, double value){
36      this.prob[posx][posy] = value;
37      }
38
39     //set probability of current position
40    public double getProb(int posx,int posy){
41      return this.prob[posx][posy];
42      }
43
44   //print all possible positions
45    public void getPosition(ArrayList<Position> possibleP){
46    for(Position p:possibleP){
47      System.out.print("["+p.getX()+","+p.getY()+"] ");
48      }
49    }
50
51   //print the whole maze
52    public void printMaze(){
53      //if count == this.height, then need to begin a new line
54      int count = 0;
55      for(int i = 0; i<this.width;i++){
56        for(int j = 0; j<this.height;j++){
57          if(this.color[i][j]==R){
58            System.out.print(" R:"+"("+i+","+j+"),"+this.getProb(i, j));
59              count++;
60              if(count==this.height){
61                count=0;
62                System.out.println();
63                }
64              }else if(this.color[i][j]==G){
65                System.out.print(" G:"+"("+i+","+j+"),"+this.getProb(i, j));
66                  count++;
67                  if(count==this.height){
68                    count=0;
69                    System.out.println();
70              }else if(this.color[i][j]==B){
71                System.out.print(" B:"+"("+i+","+j+"),"+this.getProb(i, j));
72                count++;
73                if(count==this.height){
74                  count=0;
75                  System.out.println();
76                  }
77              }else if(this.color[i][j]==Y){
78                 System.out.print(" Y:"+"("+i+","+j+"),"+this.getProb(i, j));
79                 count++;
80                 if(count==this.height){
81                   count=0;
82                   System.out.println();
83                   }
84                 }
85               }
```

```
            }
            System.out.println();
          }
      }
```

In this class, I will use 2-dimension array to represent color for each (x,y) position and when the user set color and position in the main function, then it will call setColorandPosition function, so it will both set color and position for each cell. Moreover, in the Maze class, a ArrayList<Position> p can help to store all positions in an arrayList. Also, this class contains two printer, one to print all possible position for current possible position list and the other one is to print the whole maze information. And the also helper functions are going to help to get or set probability in sensor matrix and transition matrix.

3.2 Position.java

My position class is a helper class to help me better deal with (x,y) position and to better using HashMap in transition model to represent each pair of positions have one transition probability. My code for it as shown below:

Listing 2: Position.java

```
public class Position {
  private int x;
  private int y;

  //initial new position
  public Position(int posx,int posy){
        this.x = posx;
        this.y = posy;
   }

  //set x
  public void setX(int x){
        this.x = x;
    }

  //set y
  public void setY(int y){
        this.y = y;
    }

  //get x
  public int getX(){
        return this.x;
    }
  //get y
  public int getY(){
        return this.y;
    }

  //set position using setX() and setY()
  public void setPosition(int x,int y){
        this.setX(x);
        this.setY(y);
```

```
    }
}
```

This class is very simple, it can set position for each cell and in order get each position x and y, I write getX() and getY().

3.3 SensorModel.java

Sensor model is one of the two important model for this program. It deals with each color value of the input sequence and update probabilities at each step as well as predict possible positions of the robot at each step. To update probabilities each time, it need to multiply transition matrix as well. Here is my code for my sensor model:

Listing 3: SensorModel.java

```
public class SensorModel {
  //the sequence of input colors
  private int[] readValue;
  //the transition matrix
  private TransitionModel t;

  //get input colors and transition model
  public SensorModel(int[] r,TransitionModel t){
    this.readValue = new int[r.length];
    this.readValue = r;
    this.t = t;
    }


   public void updateProb(Maze m){
     int count = 0;
     //update sensor model
     for(int i = 0;i<this.readValue.length;i++){
       for(int j = 0;j<m.width;j++){
         for(int k = 0;k<m.height;k++){
           if(m.getColor(j, k)==this.readValue[i]){
             double currentProb = m.getProb(j, k)*0.88;
             m.setProb(j, k, currentProb);
             }else{
               double currentNProb = m.getProb(j, k)*0.04;
               m.setProb(j, k, currentNProb);
               }
             }
           }

      //***************normalization***************
      double sum1 = 0.0;
      for(int ni=0;ni<m.width;ni++){
        for(int nj=0;nj<m.height;nj++){
        sum1 =sum1 + m.getProb(ni, nj);
        }
      }

```

```
39        for(int ni=0;ni<m.width;ni++){
40          for(int nj=0;nj<m.height;nj++){
41            double temp = m.getProb(ni, nj)/sum1;
42            m.setProb(ni, nj,temp);
43              }
44          }
45        //*****************************************
46
47        //print maze for each time step
48        System.out.println("T("+(count)+"), Sensor Model: ");
49        count++;
50        m.printMaze();
51
52        //*************possible position**************
53        double max0 = 0.0;
54        int maxi0 = -1;
55        int maxj0 = -1;
56        for(int mi =0;mi<m.width;mi++){
57          for (int j = 0; j <m.height;j++){
58            if( m.getProb(mi, j)>max0){
59              max0 = m.getProb(mi, j);
60              maxi0 = mi;
61              maxj0 = j;
62              }
63            }
64          }
65
66       ArrayList<Position> possibleP0 = new ArrayList<Position>();
67       for(int ti = 0;ti<m.width;ti++){
68         for(int j = 0;j<m.height;j++){
69           if(m.getProb(ti, j)==max0){
70             Position needToBeAdd = new Position(ti,j);
71             possibleP0.add(needToBeAdd);
72               }
73             }
74           }
75
76        System.out.print("Most possible position: ");
77        m.getPosition(possibleP0);
78        System.out.println();
79        System.out.println();
80        //*****************************************
81        if(i!=this.readValue.length-1){
82          double[][] tempnewprob = new double[m.width][m.height];
83          for(Position eachp: m.p){
84            ArrayList<Position> neighbors = this.t.getRelationship(eachp);
85            double temp = 0.0;
86            for(Position eachneighbors: neighbors){
87              double a = this.t.getTransitionProb(eachp, eachneighbors);
88              double b = m.getProb(eachneighbors.getX(), eachneighbors.getY());
89              temp = temp +a*b;
```

```
            }
            tempnewprob[eachp.getX()][eachp.getY()] = temp;
          }

      for(int x = 0; x<m.width;x++){
        for(int y = 0; y<m.height;y++){
          m.setProb(x,y,tempnewprob[x][y]);
            }
           }
         }
       }
     }
  }
```

In my above code, it contains two main functions, one is the constructor and the other one is updateProb. The constructor is simple, it reads the input color sequence as an array and also it needs information of the initial transition matrix, so it also reads a TransitionModel t.

The other function, updataProb, is very important one. The idea of it is: first calculate transition matrix, and update probabilities for each cell, second sensor model get probabilities for each cell and calculate sensor matrix. Since at time step 0, the transition matrix can be treated as each cell has the same probabilities(for 4X4 maze, each cell has a probability of 0.625). So, at the first iteration step, it will calculate it by multiply it by 0.88 for the right color and others by 0.04. Also, after each calculation, it needs to do normalization to make the sum all probabilities equals to one(like the code below).

Listing 4: update sensor model

```
 for(int i = 0;i<this.readValue.length;i++){
   for(int j = 0;j<m.width;j++){
      for(int k = 0;k<m.height;k++){
        if(m.getColor(j, k)==this.readValue[i]){
          double currentProb = m.getProb(j, k)*0.88;
           m.setProb(j, k, currentProb);
          }else{
            double currentNProb = m.getProb(j, k)*0.04;
            m.setProb(j, k, currentNProb);
            }
           }
        }
      //***************normalization***************
      double sum1 = 0.0;
      for(int ni=0;ni<m.width;ni++){
        for(int nj=0;nj<m.height;nj++){
        sum1 =sum1 + m.getProb(ni, nj);
        }
      }

      for(int ni=0;ni<m.width;ni++){
        for(int nj=0;nj<m.height;nj++){
          double temp = m.getProb(ni, nj)/sum1;
          m.setProb(ni, nj,temp);
            }
        }
      //*******************************************
```

And then, after the time step 0, it will attend to make a movement. So, at this time, the transition model will update by the rule: for each position, get its neighbors list. And for each neighbors, get transition probability of current position with this neighbors, and calculate the sum of all neighbors and the current position's transition probabilities. Then, update a temp probability array. After finish calculation all positions, update the maze probabilities with this temp probability array(like the code below).

Listing 5: Update transition model

```
if(i!=this.readValue.length-1){
       double[][] tempnewprob = new double[m.width][m.height];
       for(Position eachp: m.p){
         ArrayList<Position> neighbors = this.t.getRelationship(eachp);
         double temp = 0.0;
         for(Position eachneighbors: neighbors){
           double a = this.t.getTransitionProb(eachp, eachneighbors);
           double b = m.getProb(eachneighbors.getX(), eachneighbors.getY());
           temp = temp +a*b;
           }
           tempnewprob[eachp.getX()][eachp.getY()] = temp;
         }
     //tempnewprob store all new transition probabilities
     for(int x = 0; x<m.width;x++){
       for(int y = 0; y<m.height;y++){
         m.setProb(x,y,tempnewprob[x][y]);
           }
          }
        }
```

Then, the sensor model will finish it jobs with the help of transition model. So, now I will discuss about transition model.

3.4 TransitionModel.java

The transition model deals with each possible movement of the robot. At first, the robot has the same probability to be stay at each cell on the maze. However, since each cell has its color, then according to the input sequence of color, the first step of the robot will depend on the first input color. For example, if the first color is Blue, then the robot may has much more possible stand on the Blue cell at the first step(Since the sensor has a probability of 0.88 to tell truth). Then for the next step, the probability will multiply the transition matrix.

The transition matrix is fixed with a fixed maze. For example, for a 4X4 maze, each cell on the corner will has 3 neighbors. And the next step, the moving probability of the robot may stand on the same cell is 0.5(Since it has two directions to hit the wall and stand still) and the moving probability of it to move to two other legal cell is both 0.25. Then each cell inside the maze(Without nearby walls), has 4 directions to go and will not hit a wall, then all of these 4 directions has a moving probability of 0.25 and the neighbors list does not contains the same cell. For each cell that has a wall but not corner, it will also have 4 neighbors with moving probability of 0.25 but one of the neighbor is itself, the same cell.

In order to represent the transition model well, I use HashMap to help me. Like the following code(My Transition Model constructor):

Listing 6: Transition Model data structure

```
1  public class TransitionModel {
2    //neighbor
3    HashMap<Position,Position> neighbor;
4    //neighbor list
5    HashMap<Position,ArrayList<Position>> relation;
6    //probability of one position move to another position
7    HashMap<HashMap<Position,Position>, Double> transitionProb;
8    int width;
9    int height;
10   public TransitionModel(int w, int h){
11     this.relation = new HashMap<Position,ArrayList<Position>>();
12     neighbor = new HashMap<Position,Position>();
13     transitionProb = new HashMap<HashMap<Position,Position>,Double>();
14     this.width = w;
15     this.height = h;
16     }
```

To represent the relationships of transition model, I used three HashMap.

The first one is `HashMap<Position,Position> neighbor`, which records two cell that are reachable to each other.

The second one is`HashMap<Position,ArrayList<Position>> relation`, which helps to record for one position what is the neighbors list of it.

The third one is `HashMap<HashMap<Position,Position>, Double> transitionProb`, which help to record the transition probability(moving probability) for one position to another position.

In all, these three HashMap works well for my program.

Then, to create the transition model, I should first have the relationship of each cell, in other words, get neighbors for each cell. So my code for get neighbors is as below:

Listing 7: get neighbors

```
1    //set neighbor list for position a
2    public void setRelationship(Position a, ArrayList<Position> b){
3      this.relation.put(a, b);
4      }
```

```
5
6   //get neighbors for each position in posList
7   public TransitionModel setTransitionMatrix(ArrayList<Position>
        posList,TransitionModel t){
8   for(Position eachp: posList){
9     if(eachp.getX()!=0&&eachp.getX()!=this.width
10       &&eachp.getY()!=0&&eachp.getY()!=this.height){
11      ArrayList<Position> recordneighbors = new ArrayList<Position>();
12      for(Position eachlittlep:posList){
13        if(eachlittlep.getX()==eachp.getX()+1 && eachlittlep.getY() ==
              eachp.getY()
14          ||eachlittlep.getX()==eachp.getX()-1 && eachlittlep.getY() ==
              eachp.getY()
15          ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
              eachp.getY()+1
16          ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
              eachp.getY()-1){
17          recordneighbors.add(eachlittlep);
18            }
19           }
20       t.setRelationship(eachp, recordneighbors);
21       }else if(eachp.getX()==0&&eachp.getY()==0
22                   ||eachp.getX()==0&&eachp.getY()==this.width
23                   ||eachp.getX()==this.height&&eachp.getY()==0
24                   ||eachp.getX()==this.width&&eachp.getY()==this.height){
25        if(eachp.getX()==0&&eachp.getY()==0){
26          ArrayList<Position> recordneighbors = new ArrayList<Position>();
27          for(Position eachlittlep:posList){
28            if(eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                  eachp.getY()
29              ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                  eachp.getY()+1
30              ||eachlittlep.getX()==eachp.getX()+1 && eachlittlep.getY() ==
                  eachp.getY()){
31                recordneighbors.add(eachlittlep);
32                 }
33               }
34             t.setRelationship(eachp, recordneighbors);
35           }
36
37      if(eachp.getX()==0&&eachp.getY()==this.width){
38        ArrayList<Position> recordneighbors = new ArrayList<Position>();
39       for(Position eachlittlep:posList){
40         if(eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
               eachp.getY()
41           ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
               eachp.getY()-1
42           ||eachlittlep.getX()==eachp.getX()+1 && eachlittlep.getY() ==
               eachp.getY()){
43           recordneighbors.add(eachlittlep);
44             }
```

```
45              }
46              t.setRelationship(eachp, recordneighbors);
47            }
48
49          if(eachp.getX()==this.height&&eachp.getY()==0){
50            ArrayList<Position> recordneighbors = new ArrayList<Position>();
51            for(Position eachlittlep:posList){
52              if(eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                    eachp.getY()
53               ||eachlittlep.getX()==eachp.getX()-1 && eachlittlep.getY() ==
                    eachp.getY()
54               ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                    eachp.getY()+1){
55               recordneighbors.add(eachlittlep);
56                }
57              }
58             t.setRelationship(eachp, recordneighbors);
59             }
60
61            if(eachp.getX()==this.width&&eachp.getY()==this.height){
62              ArrayList<Position> recordneighbors = new ArrayList<Position>();
63              for(Position eachlittlep:posList){
64                if(eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                      eachp.getY()
65                ||eachlittlep.getX()==eachp.getX()-1 && eachlittlep.getY() ==
                      eachp.getY()
66                ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                      eachp.getY()-1){
67                recordneighbors.add(eachlittlep);
68                }
69              }
70            t.setRelationship(eachp, recordneighbors);
71             }
72            }else if(eachp.getY()==0){
73              ArrayList<Position> recordneighbors = new ArrayList<Position>();
74              for(Position eachlittlep:posList){
75                if(eachlittlep.getX()==eachp.getX()-1 && eachlittlep.getY() ==
                      eachp.getY()
76                 ||eachlittlep.getX()==eachp.getX()+1 && eachlittlep.getY() ==
                       eachp.getY()
77                 ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                       eachp.getY()+1
78                 ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                       eachp.getY()){
79                  recordneighbors.add(eachlittlep);
80                  }
81                }
82
83                t.setRelationship(eachp, recordneighbors);
84            }else if(eachp.getY()==this.width){
85              ArrayList<Position> recordneighbors = new ArrayList<Position>();
```

```
86              for(Position eachlittlep:posList){
87               if(eachlittlep.getX()==eachp.getX()-1 && eachlittlep.getY() ==
                     eachp.getY()
88                ||eachlittlep.getX()==eachp.getX()+1 && eachlittlep.getY() ==
                     eachp.getY()
89                ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                     eachp.getY()-1
90                ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                     eachp.getY()){
91                 recordneighbors.add(eachlittlep);
92                  }
93                 }
94              t.setRelationship(eachp, recordneighbors);
95         }else if(eachp.getX()==0){
96              ArrayList<Position> recordneighbors = new ArrayList<Position>();
97               for(Position eachlittlep:posList){
98                 if(eachlittlep.getX()==eachp.getX()+1 && eachlittlep.getY() ==
                     eachp.getY()
99                  ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                        eachp.getY()-1
100                 ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                        eachp.getY()+1
101                 ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                        eachp.getY()){
102                  recordneighbors.add(eachlittlep);
103                   }
104                 }
105              t.setRelationship(eachp, recordneighbors);
106        }else if(eachp.getX()==this.height){
107            ArrayList<Position> recordneighbors = new ArrayList<Position>();
108             for(Position eachlittlep:posList){
109               if(eachlittlep.getX()==eachp.getX()-1 && eachlittlep.getY() ==
                     eachp.getY()
110                ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                        eachp.getY()-1
111                ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                        eachp.getY()+1
112                ||eachlittlep.getX()==eachp.getX() && eachlittlep.getY() ==
                        eachp.getY()){
113                recordneighbors.add(eachlittlep);
114                }
115               }
116             t.setRelationship(eachp, recordneighbors);
117              }
118            }
119            return t;
120        }
```

My idea of get neighbors is to separate the maze into 4 possible positions, one is the positions are on the corner, the other one is the position is inside the maze without walls, and the other one is has wall on top or bottom, the final one is has wall on left or right.

Then, after finishing setting neighbor list for each position on the maze. I need to set transition probability

for each pair of neighbor. The code of it is as followed:

Listing 8: set transition probability

```
1   //set transition probability for position a and position b
2   public void setTransitionProb(Position a, Position b, double value){
3     this.neighbor.put(a, b);
4     HashMap<Position,Position> current = new HashMap<Position,Position>();
5     current.put(a, b);
6     this.transitionProb.put(current, value);
7     }
8
9   //set transition probability for all cell on maze m
10  public TransitionModel CalculateTransitionMatrix(Maze m,TransitionModel t){
11    ArrayList<Position> p = m.p;
12    for(Position eachp:p){
13     ArrayList<Position> neighbors = t.getRelationship(eachp);
14     if(neighbors.contains(eachp)){
15       if(neighbors.size()==3){
16         for(Position eachneighbor: neighbors)
17           if(eachneighbor.getX()==eachp.getX()
18              &&eachneighbor.getY()==eachp.getY()){
19             t.setTransitionProb(eachp, eachneighbor, 0.5);
20             }else{
21              t.setTransitionProb(eachp, eachneighbor, 0.25);
22              }
23             }else if(neighbors.size()==4){
24               for(Position eachneighbor: neighbors){
25               t.setTransitionProb(eachp, eachneighbor,0.25);
26               }
27             }
28           }else{
29             for(Position eachneighbor: neighbors){
30               t.setTransitionProb(eachp, eachneighbor,0.25);
31                }
32              }
33            }
34             return t;
35      }
```

My idea of set the transition probability for each position is to consider about the number of neighbors for the current position as well as if the neighbors list of the current position contains the current position or not.

3.5 Driver.java

Now, all my functional classes are all discussed, then finally, I will show my main function, the Driver of this program. The driver simply set the size of the maze and colors of each position. My code is as below:

Listing 9: Driver.java

```
public class Driver {
        public static int R = 0;
        public static int G = 1;
        public static int B = 2;
        public static int Y = 3;
        public static void main(String[] args)
        {
          //create maze
          Maze m = new Maze(4,4);

          m.setColorandPosition(0, 0, G);
          m.setColorandPosition(0, 1, R);
          m.setColorandPosition(0, 2, Y);
          m.setColorandPosition(0, 3, B);
          m.setColorandPosition(1, 0, Y);
          m.setColorandPosition(1, 1, G);
          m.setColorandPosition(1, 2, R);
          m.setColorandPosition(1, 3, Y);
          m.setColorandPosition(2, 0, G);
          m.setColorandPosition(2, 1, Y);
          m.setColorandPosition(2, 2, B);
          m.setColorandPosition(2, 3, R);
          m.setColorandPosition(3, 0, R);
          m.setColorandPosition(3, 1, B);
          m.setColorandPosition(3, 2, Y);
          m.setColorandPosition(3, 3, B);
          System.out.println("The original maze is: ");
          m.printMaze();
          System.out.println("*****************************");

          //create transition model
          TransitionModel t = new TransitionModel(4,4);
          t = t.setTransitionMatrix(m.p, t);
          t = t.CalculateTransitionMatrix(m, t);

          //create sensor model
          int [] sequence = {Y,G};
          SensorModel s = new SensorModel(sequence,t);
          s.updateProb(m);
```

Then, by running Driver.java, I can test my program.

4 Test Result

In my homework, I did not using fancy maze to show my result. But I think my result of ASCII is clear enough to show my works. In each my test case, I will firstly print out the original maze information, and then follow up each result of each time step containing probability for each cell as well as possible location of the robot.

First I will show my original maze is:

Listing 10: Maze

```
G:(0,0) R:(0,1) Y:(0,2) B:(0,3)
Y:(1,0) G:(1,1) R:(1,2) Y:(1,3)
G:(2,0) Y:(2,1) B:(2,2) R:(2,3)
R:(3,0) B:(3,1) Y:(3,2) B:(3,3)
```

So the easiest and clear test case is to test a sequence of only one color, let me test {Y} and let's see what is the result:

```
The original maze is:
 G:(0,0),0.0625 R:(0,1),0.0625 Y:(0,2),0.0625 B:(0,3),0.0625
 Y:(1,0),0.0625 G:(1,1),0.0625 R:(1,2),0.0625 Y:(1,3),0.0625
 G:(2,0),0.0625 Y:(2,1),0.0625 B:(2,2),0.0625 R:(2,3),0.0625
 R:(3,0),0.0625 B:(3,1),0.0625 Y:(3,2),0.0625 B:(3,3),0.0625

******************************************************
T(0), Sensor Model:
 G:(0,0),0.008264462809917356 R:(0,1),0.008264462809917356 Y:(0,2),0.18181818181818182 B:(0,3),0.008264462809917356
 Y:(1,0),0.18181818181818182 G:(1,1),0.008264462809917356 R:(1,2),0.008264462809917356 Y:(1,3),0.18181818181818182
 G:(2,0),0.008264462809917356 Y:(2,1),0.18181818181818182 B:(2,2),0.008264462809917356 R:(2,3),0.008264462809917356
 R:(3,0),0.008264462809917356 B:(3,1),0.008264462809917356 Y:(3,2),0.18181818181818182 B:(3,3),0.008264462809917356

Most possible position: [0,2] [1,0] [1,3] [2,1] [3,2]
```

Figure 3: Y

The result is clear, and get all positions that are colored "Y".

Then for a test case of input sequence of {Y, G}, the result is as followed: Since the first color is "Y",

```
The original maze is:
 G:(0,0),0.0625 R:(0,1),0.0625 Y:(0,2),0.0625 B:(0,3),0.0625
 Y:(1,0),0.0625 G:(1,1),0.0625 R:(1,2),0.0625 Y:(1,3),0.0625
 G:(2,0),0.0625 Y:(2,1),0.0625 B:(2,2),0.0625 R:(2,3),0.0625
 R:(3,0),0.0625 B:(3,1),0.0625 Y:(3,2),0.0625 B:(3,3),0.0625

******************************************************
T(0), Sensor Model:
 G:(0,0),0.008264462809917356 R:(0,1),0.008264462809917356 Y:(0,2),0.18181818181818182 B:(0,3),0.008264462809917356
 Y:(1,0),0.18181818181818182 G:(1,1),0.008264462809917356 R:(1,2),0.008264462809917356 Y:(1,3),0.18181818181818182
 G:(2,0),0.008264462809917356 Y:(2,1),0.18181818181818182 B:(2,2),0.008264462809917356 R:(2,3),0.008264462809917356
 R:(3,0),0.008264462809917356 B:(3,1),0.008264462809917356 Y:(3,2),0.18181818181818182 B:(3,3),0.008264462809917356

Most possible position: [0,2] [1,0] [1,3] [2,1] [3,2]

T(1), Sensor Model:
 G:(0,0),0.19011406844106463 R:(0,1),0.008641548565502939 Y:(0,2),0.008641548565502939 B:(0,3),0.015900449360525405
 Y:(1,0),0.008641548565502939 G:(1,1),0.34980988593155893 R:(1,2),0.015900449360525405 Y:(1,3),0.0010369858278603526
 G:(2,0),0.34980988593155893 Y:(2,1),0.00138264777048047 B:(2,2),0.015900449360525405 R:(2,3),0.00829588662288282
 R:(3,0),0.00138264777048047 B:(3,1),0.015554787417905287 Y:(3,2),0.0010369858278603526 B:(3,3),0.007950224680262703

Most possible position: [1,1] [2,0]
```

Figure 4: Y,G

the time step of 0 calculate the most possible positions are: [0,2] [1,0] [1,3] [2,1] [3,2], which are all the cells of color "Y". And since the second color is "G", in the original maze. There are there cells that are colored "G": [0,0], [1,1], [2,0]. Since [0,0] has three neighbors, one is it self, another one is a [1,0] with color "Y", and the other one is [0,1] with color "R". So for [0,0], it can only has one possibility that are moved from cell with color "Y". While position [1,1] and [2,0] all have four neighbors, and two neighbors are cells with color "Y". Therefore, at time step 1, there are two possible positions, which are [1,1] and [2,0]. Thus, my program runs right.

Also I can prove my works well by adding one input "R" which makes my input sequence to be {Y, G, R}, then my result is: From this test case, although in my original maze there are three possible paths for

```
The original maze is:
 G:(0,0),0.0625 R:(0,1),0.0625 Y:(0,2),0.0625 B:(0,3),0.0625
 Y:(1,0),0.0625 G:(1,1),0.0625 R:(1,2),0.0625 Y:(1,3),0.0625
 G:(2,0),0.0625 Y:(2,1),0.0625 B:(2,2),0.0625 R:(2,3),0.0625
 R:(3,0),0.0625 B:(3,1),0.0625 Y:(3,2),0.0625 B:(3,3),0.0625

******************************************************
T(0), Sensor Model:
 G:(0,0),0.008264462809917356 R:(0,1),0.008264462809917356 Y:(0,2),0.18181818181818182 B:(0,3),0.008264462809917356
 Y:(1,0),0.18181818181818182 G:(1,1),0.008264462809917356 R:(1,2),0.008264462809917356 Y:(1,3),0.18181818181818182
 G:(2,0),0.008264462809917356 Y:(2,1),0.18181818181818182 B:(2,2),0.008264462809917356 R:(2,3),0.008264462809917356
 R:(3,0),0.008264462809917356 B:(3,1),0.008264462809917356 Y:(3,2),0.18181818181818182 B:(3,3),0.008264462809917356

Most possible position: [0,2] [1,0] [1,3] [2,1] [3,2]

T(1), Sensor Model:
 G:(0,0),0.19011406844106463 R:(0,1),0.008641548565502939 Y:(0,2),0.008641548565502939 B:(0,3),0.015900449360525405
 Y:(1,0),0.008641548565502939 G:(1,1),0.34980988593155893 R:(1,2),0.015900449360525405 Y:(1,3),0.0010369858278603526
 G:(2,0),0.34980988593155893 Y:(2,1),0.00138264777048047 B:(2,2),0.015900449360525405 R:(2,3),0.00829588662288282
 R:(3,0),0.00138264777048047 B:(3,1),0.015554787417905287 Y:(3,2),0.0010369858278603526 B:(3,3),0.007950224680262703

Most possible position: [1,1] [2,0]

T(2), Sensor Model:
 G:(0,0),0.012501902463418346 R:(0,1),0.385536929532755 Y:(0,2),0.0015437131737438307 B:(0,3),0.0013045463440088708
 Y:(1,0),0.02825429956732546 G:(1,1),0.0010871219533407257 R:(1,2),0.2597351770921662 Y:(1,3),0.0012610614658752418
 G:(2,0),0.011360424412410584 Y:(2,1),0.02299262931315635 B:(2,2),8.370839040723589E-4 R:(2,3),0.017220011740917095
 R:(3,0),0.25471267366773204 B:(3,1),1.1958341486747983E-4 Y:(3,2),0.0012393190268084272 B:(3,3),2.9352292740199596E-4

Most possible position: [0,1]
```

Figure 5: Y,G,R

{Y, G, R}, regarding to filtering algorithm that I have used, it filtered those two impossible path like what I have discussed in the previous test case, then it find the only possible position of the final input "R" and it is position [0,1].

5 Citation

[1].http://stackoverflow.com/questions/17487356/hidden-markov-model-for-multiple-observed-variables