updateProbabilities:

```
// TODO: update the probabilities of where the AI thinks it is based on the action selected and the new sonar readings
//       To do this, you should update the 2D-array "probs"
// Note: sonars is a bit string with four characters, specifying the sonar reading in the direction of North, South, East, and West
//       For example, the sonar string 1001, specifies that the sonars found a wall in the North and West directions, but not in the South and East directions
void updateProbabilities(int action, String sonars) {
    // your code
    transitionModel(action);
    sensorModel(sonars);
    normalize();

    myMaps.updateProbs(probs); // call this function after updating your probabilities so that the
                               //  new probabilities will show up in the probability map on the GUI
}
```

Transition Model:

```
void transitionModel(int action){
    double incorrectMoveProb = (1 - moveProb)/4;
    double[][] transitionProb = new double[probs.length][probs[0].length];

    for(int i = 1; i < mundo.height; i++){
        for(int j = 1; j < mundo.width; j++){
            List<Position> neighbors = getNeighbors(i, j);
            neighbors.add(new Position(i, j, STAY));

            for(Position pos : neighbors){
                if(mundo.grid[pos.x][pos.y] == 1 && pos.action == action){
                    transitionProb[i][j] += probs[i][j] * moveProb;
                }
                else if(mundo.grid[pos.x][pos.y] == 1){
                    transitionProb[i][j] += probs[i][j] * incorrectMoveProb;
                }
                else if (pos.action == action) {
                    transitionProb[pos.x][pos.y] += probs[i][j] * moveProb;
                }
                else {
                    transitionProb[pos.x][pos.y] += probs[i][j] * incorrectMoveProb;
                }

            }
        }
    }
    probs = transitionProb;
}
```

Sensor Model:

```
void sensorModel(String sonars){
    double sensorInaccuracy = 1 - sensorAccuracy;

    for(int i = 1; i < mundo.height; i++) {
        for (int j = 1; j < mundo.width; j++) {
            if (mundo.grid[i][j] != 0) {
                probs[i][j] = 0.0;
                continue;
            }

            String correctSensor = correctSensor(i, j);
            double currentProb = probs[i][j];

            for(int q = 0; q < correctSensor.length(); q++){
                if(correctSensor.charAt(q) == sonars.charAt(q)){
                    currentProb = currentProb * sensorAccuracy;
                }
                else {
                    currentProb = currentProb * sensorInaccuracy;
                }
            }
            probs[i][j] = currentProb;
        }
    }
}
```

normalize:

```
void normalize(){
    double sum = 0.0;

    for(int i = 1; i < mundo.height; i++) {
        for (int j = 1; j < mundo.width; j++) {
            if (mundo.grid[i][j] == 1) {
                continue;
            }
            sum += probs[i][j];
        }
    }

    for(int i = 1; i < mundo.height; i++) {
        for (int j = 1; j < mundo.width; j++) {
            if (mundo.grid[i][j] == 1) {
                continue;
            }
            probs[i][j] = probs[i][j]/sum;
        }
    }
}
```

correctSensor:

```
String correctSensor(int i, int j){
    StringBuilder sb = new StringBuilder();
    List<Position> neighbors = getNeighbors(i, j);

    for(Position pos: neighbors){
        switch (pos.action) {
            case NORTH:
                if(mundo.grid[pos.x][pos.y] == 1){
                    sb.append("1");
                }
                else {
                    sb.append('0');
                }
                break;
            case SOUTH:
                if(mundo.grid[pos.x][pos.y] == 1){
                    sb.append("1");
                }
                else {
                    sb.append('0');
                }
                break;
            case EAST:
                if(mundo.grid[pos.x][pos.y] == 1){
                    sb.append("1");
                }
                else {
                    sb.append('0');
                }
                break;
            case WEST:
                if(mundo.grid[pos.x][pos.y] == 1){
                    sb.append("1");
                }
                else {
                    sb.append('0');
                }
                break;
        }
    }
```

getNeighbors:

```
final class Position {
    public int x;
    public int y;
    public int action;
    public Position(int x, int y, int action){
        this.x = x;
        this.y = y;
        this.action = action;
    }
}

List<Position> getNeighbors(int i, int j){
    List<Position> neighbors = new ArrayList<>();
    if(j != 0) {
        neighbors.add(new Position(i, j-1, NORTH));
    }
    if(j != mundo.height - 1) {
        neighbors.add(new Position(i, j+1, SOUTH));
    }
    if(i != mundo.width - 1){
        neighbors.add(new Position(i+1, j, EAST));
    }
    if(i != 0) {
        neighbors.add(new Position(i-1, j, WEST));
    }

    return neighbors;
}
```

Advice to a younger me:

While the assignment was pretty clear, the thing that will stump you are getting the neighbors of each cell. Because your transition model relies on the squares around it, don't try and update each cell when you reach it, but rather update it's neighboring cells. This way you won't get confused on directions. Walls also need to be taken into account, and if you are not careful and diagram it out on paper before coding it, you will refactor the motion model over and over again like I did. One other thing that screwed me up, was make sure your neighbors are inserted into the Positions array in the right order, or you will think that you are getting correct sensor readings when really you are not. How I started to debug this was first I just used the transition model with a 1.0 probability, and made sure those results made sense on the screen. Then I used the sensor model with a 1.0 probability. This is a good way to start testing, as you don't have to account for incorrect readings. Also make sure that when you start, know that the sensor model just needs to update the values you get from the transition model, and that you don't need to multiply it again back into the probability matrix.

I would say the biggest pitfalls you will run into are not handling neighbors (checking for walls and behavior if you are running into a wall, etc.) and assuming you get correct sensor readings from your function. Just stay on top of those and this assignment will be much easier.