Probabilistic Reasoning Over Time

Michelle Shu

March 6, 2014

1 Introduction

In some scenarios, agents are unable to obtain complete information about their environments and must piece together their current state based on experience and partial observations. In this paper, I will describe the application of a probabilistic temporal model to a specific problem of robot localization.

The premise of the problem is this: A robot is located at an unknown location on an nxn grid. The robot can move in any of the 4 cardinal directions (North, East, South, West), but cannot detect the result of its motion or the direction it moved. Thus, if the robot tries to move into a wall, it will not move and stay in its original position, but it has no means of knowing whether it has just moved or not. The robot has only one sensor. It points downward and can tell the robot with better than chance accuracy the color of the grid square beneath it. Using this observation, along with an internal knowledge of both the map and the colors of its grid squares, the robot must infer a probabilistic distribution of its current location.

2 A Temporal Model for the Blind Robot Problem

An agent in an unknown environment must keep track what it knows about its current state as it explores the world. Our robot keeps track of a weighted belief state, reflecting the locations where it could possibly be at the current time point and corresponding ratings for its confidence of being in each location (likeliness of being in each location). Using probabilistic inference, the robot modifies its belief state at each discrete time point by following a transition model and a sensor model. The transition model specifies how the robot's location in the world changes with each movement. The sensor model specifies how the color of the floor at a given location is read from the sensor.

2.1 Transition Model

I model the changes in the robot's belief state as a first-order Markov process, in which the current belief state (at time t) is derived only from the information contained in the previous belief state (time $t-1$), without consideration of the earlier states that occurred before $t-1$. Figure 1 depicts a simple case of a robot on a 2x2 grid. At time $t-1$, the robot believes with some set of probabilities that it is either at location 0, 1, 2 or 3. Let $P(L_{t-1}=0)$ denote what the robot believes to be its probability of being at location 0 during time $t-1$.

In each time step, the robot moves one step in a random direction, but it is not aware of which direction it moves. All possible movements are shown in Fig.1b. It assumes that the probabilities of the direction of the last move being N, E, S, W are equivalent (all 0.25). We use the law of conditional probability on the space of possible transitions to compute the values $P(L_t)$ for all grid locations. Computing these values for each grid square allows us to model the transition of the robot location from one time step to the next. For example, here is the computation for $P(L_t=0)$:

Let l_{t-1} represent all possible locations at time $t-1$.

$$P(L_t = 0) = \sum_{l_{t-1}} P(L_t = 0|l_{t-1})P(l_{t-1})$$

$$= P(L_t = 0|L_{t-1} = 0)P(L_{t-1} = 0) + P(L_t = 0|L_{t-1} = 1)P(L_{t-1} = 1) +$$
$$P(L_t = 0|L_{t-1} = 2)P(L_{t-1} = 2) + P(L_t = 0|L_{t-1} = 3)P(L_{t-1} = 3)$$

$$P(L_t = 0) = (0.5) * P(L_{t-1} = 0) + (0.25) * P(L_{t-1} = 1) + (0.25) * P(L_{t-1} = 2)$$

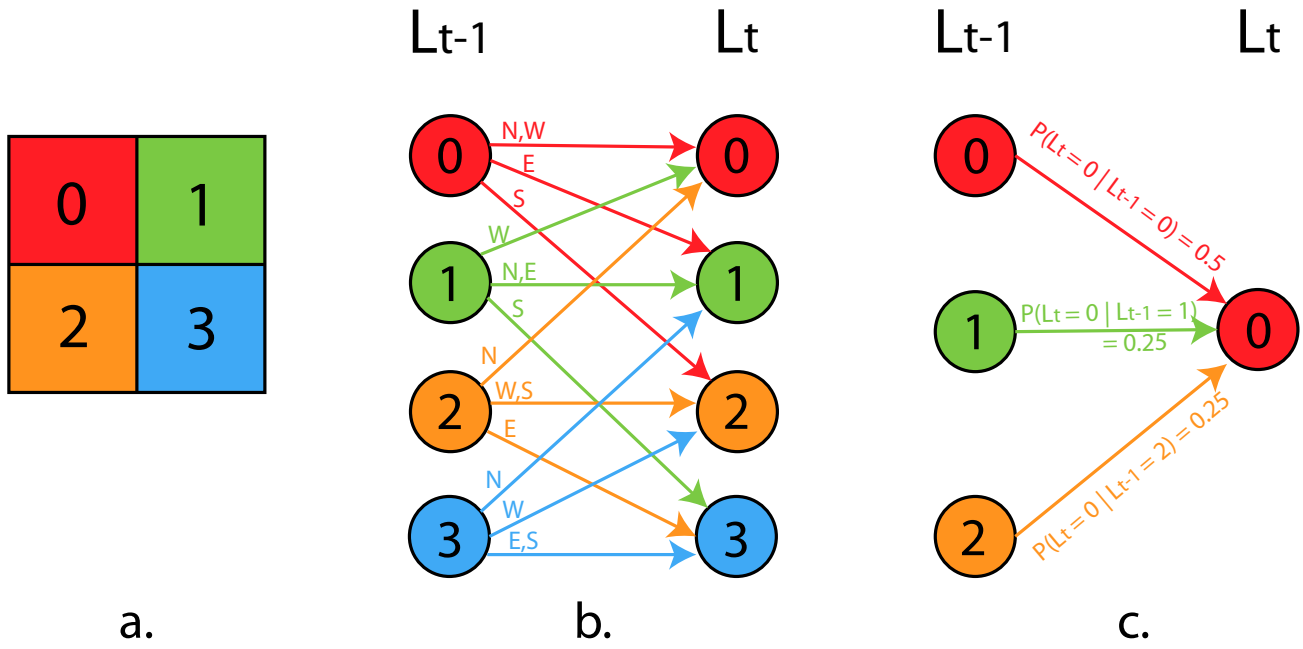


Figure 1: **State Transition Diagram** a. A 2 x 2 colored grid on which the robot resides b. All possible state transitions with movements in any of 4 directions c. The conditional probabilities from all origins of movement to position 0 at time t

2.2 Sensor Model

We declare that each grid square can be one of four colors: Red, Green, Blue or Yellow. The robot's sensor is able to read the correct color 88% of the time and reads each of the other colors (the wrong ones) 4% of the time. So if the robot is currently above a blue square, the sensor would read blue with probability 0.88, green with probability 0.04, yellow with probability 0.04 and red with probability 0.04. The sensor reading at time t provides evidence e_t to the robot which, along with the transition model probabilities, can help it infer where it is.

The sensor evidence is a consequence of the state transition model. It is the robot's location in the world that causes the sensors to pick up the reading that they do. But the robot has no way to estimate $P(L_t|C_t)$, where C_t is the color read by the sensor at time t. However, it can use what it knows about the accuracy of the sensor to estimate $P(C_t|L_t)$ which, turns out to be useful to know for our Bayesian temporal model. Specifically, $P(C_t|L_t) = 0.88$ if the color read matches the color of the grid square and $P(C_t|L_t) = 0.04$ if they do not match.

3 Probabilistic Filtering

One inference technique performed on temporal probabilistic models is called **filtering**. Filtering is the process of computing a belief state based on all evidence and experience gathered up until the current time point. I implement a filtering algorithm that maintains a belief state consisting of the estimated probabilities of the robot being in each possible position on the grid. The belief state will be incrementally updated as the robot moves by applying the transition model and the sensor model in conjunction.

Here is the structure of the filtering model (based on the figure in Russell and Norvig), which shows causal relationships between states and evidence:

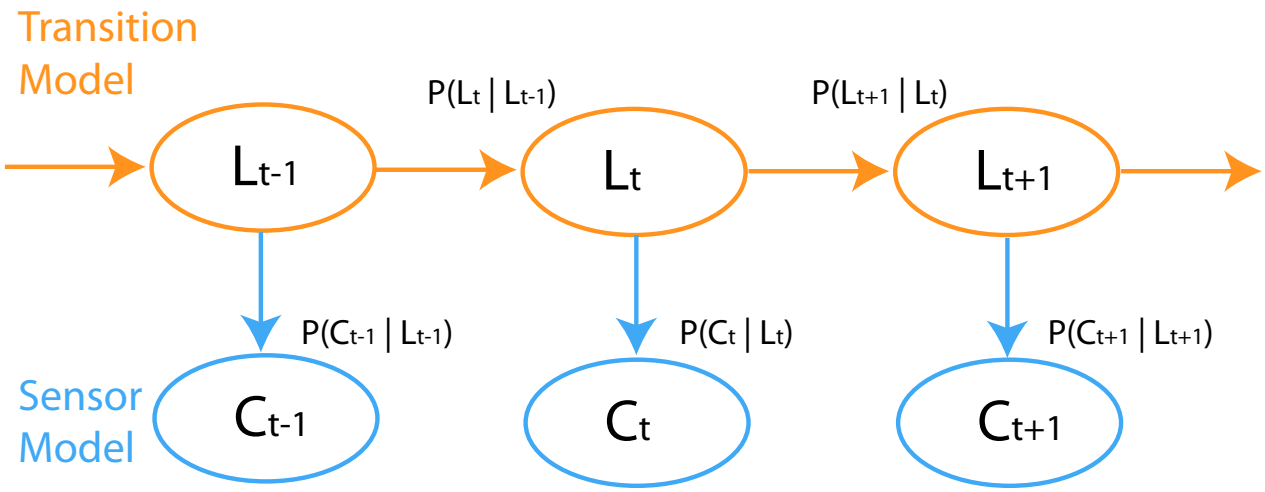


Figure 2: **Bayesian Network Structure** The relationship between the transition model and sensor model

The filter produces a conditional distribution of the robot's location at time t given the prior distribution and the evidence observed in prior steps. In other words, we define the following update step (Russell and Norvig) derived from Bayes' rule and the Markov assumption to compute the values $P(L_t|C_{1:t})$:

$$P(L_t|C_{1:t}) = \alpha P(C_t|L_t) \sum_{l_{t-1}} P(L_t|l_{t-1}) P(l_{t-1}|C_{1:t-1})$$

The factor inside the summation is derived from applying the transition model and the factor outside is derived from the sensor model. α is a normalization factor which ensures that the probabilities sum to 1.

4 Code Design

I partitioned my code into the following classes:

- `GridWorld.java`: Contains layout and colors of grid space that robot is on, and functions used to read this in from a text file format.
- `Sensor.java`: The sensor collects readings of grid square colors from the world that the robot is in. It knows the ground truth location and movements of the robot in order to produce readings, but this is information hidden from the robot itself.
- `Robot.java`: Stores belief state of robot and conducts updates using transition and sensor models.
- `Matrix.java`: Utility class containing general implementations of matrix operations (e.g. normalizing)

- `WorldView.java`: A JavaFX view that shows the robot's true location and belief state. It also allows us to step through movements and belief state updates with button callback.

5 Implementation

5.1 Grid Representation (GridWorld)

Grid layouts are read in from a text file of comma-delimited values presented in the same orientation that they would be laid out as grid squares. Each value can be any color value from the set {R, G, B, Y} or a period (.) to represent an obstacle. The example 5x5 grid I will be using is represented in text format as the following:

R,G,B,Y,B
.,R,R,G,G
.,.,B,.,Y
Y,Y,B,.,R
B,G,G,.,Y

This is read in and stored as a 2-dimensional char array by `loadGraphFromFile` in `GridWorld.java`, while also tallying the number of non-obstacle spaces.

```
private void loadGridFromFile(String filename, int w, int h) {
  this.width = w;
  this.height = h;
  this.colors = new char[height][width];
  this.validSquares = 0;

  BufferedReader br = null;
  try {
    br = new BufferedReader(new FileReader(filename));
    String[] row;
    for (int i = 0; i < height; i++) {
      row = br.readLine().split(",");
      for (int j = 0; j < width; j++) {
        colors[i][j] = row[j].trim().toCharArray()[0];
        if (! isObstacle(i, j)) {
          validSquares++;
        }
      }
    }
  } catch (IOException e) {
    e.printStackTrace();
  } finally {
    try {
      if (br != null) br.close();
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

`GridWorld.java` also includes some auxiliary functions that are used to determine whether a specific location on the grid is in bounds or an obstacle and to get the locations of the neighbors of any grid cell. The `getAdjacentLocs` function handles adjacent walls and obstacles by using the current location itself as a neighbor in these instances. This reflects the robot's behavior of staying in the same spot when it tries to move into an invalid location.

```
public boolean isObstacle(int i, int j) {
  return colors[i][j] == '.';
}

public boolean inBounds(int i, int j) {
  return !(i < 0 || j < 0 || i >= height || j >= width);
}

public int[][] getAdjacentLocs(int i, int j) {
  // Get the 4 adjacent locations to (i, j) from directions N, E, S, W
  // If the neighbor is edge of board or obstacle, replace with (i, j)
  int[][] adjacentLocs = new int[4][2];
  int index = 0;
  for (int[] dir: directions) {
    if (inBounds(i + dir[0], j + dir[1]) && ! isObstacle(i + dir[0], j + dir
    [1])) {
      adjacentLocs[index][0] = i + dir[0];
      adjacentLocs[index][1] = j + dir[1];
    } else {
      adjacentLocs[index][0] = i;
      adjacentLocs[index][1] = j;
    }
    index++;
  }
  return adjacentLocs;
}
```

5.2 Sensor

The `Sensor` class simulates the operation of an actual sensor of the robot. It collects readings from the actual world, so this class is "aware" of the robot's real location. When the robot collects the readings from the sensor, it will only receive a color with no additional information, and that color will only be accurate 88% or P_CORRECT% of the time. Each time `getNextReading` in `Sensor.java` is called by the robot, two things will happen: the robot will move in a random direction and the result of an imperfect sensor reading from the new location will be returned.

```
public Sensor(GridWorld g) {
  this.grid = g;
  this.rand = new Random();
  // Start the robot out in a random position on the board
  this.x = rand.nextInt(grid.width);
  this.y = rand.nextInt(grid.height);
}
```

```
// Get imperfect sensor reading
public char getNextReading() {
  move();
  char correctReading = grid.colors[y][x];
  if (rand.nextDouble() < P_CORRECT) {
    // Return the correct color certain percentage of the time
    return correctReading;
  } else {
    char wrongReading = colors[rand.nextInt(colors.length)];
    while (wrongReading == correctReading) {
      wrongReading = colors[rand.nextInt(colors.length)];
    }
    return wrongReading;
  }
}

// Move the robot in a random direction
public void move() {
  int[] dir = directions[rand.nextInt(directions.length)];
  if (grid.inBounds(y + dir[0], x + dir[1]) &&
      ! grid.isObstacle(y + dir[0], x + dir[1])) {
    y += dir[0];
    x += dir[1];
  }
}
```

5.3 Filtering Algorithm (Robot)

The `Robot.java` class maintains the robot's probabilistic belief state. When the `update` method is called on the robot, it gets another sensor reading and updates revises its probability distribution of its current location. To accomplish this, the robot first computes the transition update on its prior belief state and then modifies this by multiplying the sensor update.

```
public Robot(GridWorld g) {
  this.grid = g;
  this.sensor = new Sensor(grid);
  this.belief = new double[grid.height][grid.width];
  // Initialize belief state with uniform probabilities
  Matrix.setAll(belief, grid, 1.0 / (grid.validSquares));
}

public void update() {
  // Move robot and detect next color from sensor
  char color = sensor.getNextReading();
  // Update belief state
  belief = getSensorUpdate(color, getTransitionUpdate());
}
```

```
private double[][] getTransitionUpdate() {
  double[][] newBelief = new double[grid.height][grid.width];
  for (int i = 0; i < grid.height; i++) {
    for (int j = 0; j < grid.width; j++) {
      if (! grid.isObstacle(i, j)) {
        int[][] adjacentLocs = grid.getAdjacentLocs(i, j);
        newBelief[i][j] = Matrix.getAverage(belief, adjacentLocs);
      } else {
        newBelief[i][j] = 0;
      }
    }
  }
  return newBelief;
}

// Add sensor evidence factor to transition update
private double[][] getSensorUpdate(char color, double[][] transitionUpdate) {
  double[][] newBelief = new double[grid.height][grid.width];
  for (int i = 0; i < grid.height; i++) {
    for (int j = 0; j < grid.width; j++) {
      if (! grid.isObstacle(i, j)) {
        if (grid.colors[i][j] == color) {
          newBelief[i][j] = transitionUpdate[i][j] * Sensor.P_CORRECT;
        } else {
          newBelief[i][j] = transitionUpdate[i][j] * Sensor.P_INCORRECT;
        }
      } else {
        newBelief[i][j] = 0;
      }
    }
  }
  Matrix.normalize(newBelief);
  return newBelief;
}
```

The robot update functions rely on two helper functions from the matrix class. The `getAverage` function takes the average of the values in given locations in the matrix. This is used to compute the sum of conditional probabilities used in the transition model (essentially the average of probabilities in 4 locations returned by `getAdjacentLocs`).

```
public static double getAverage(double[][] matrix, int[][] locs) {
  double sum = 0.0;
  for (int l = 0; l < locs.length; l++) {
    int[] loc = locs[l];
    sum += matrix[loc[0]][loc[1]];
  }
  return (sum / (double) locs.length);
}
```

The `normalize` method normalizes the probability values in the belief state matrix so that they sum to 1.

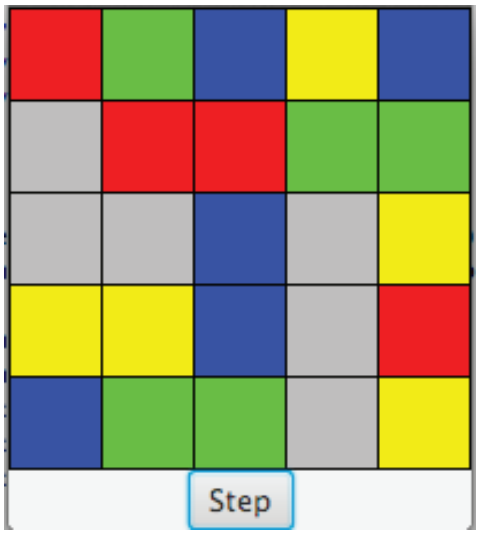
```
public static void normalize(double[][] matrix) {
  // First pass: take sum of all values in matrix
  double sum = 0;
  for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix.length; j++) {
      sum += matrix[i][j];
    }
  }
  // Second pass: divide all values by sum
  for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix.length; j++) {
      matrix[i][j] /= sum;
    }
  }
}
```

6 Sample Output

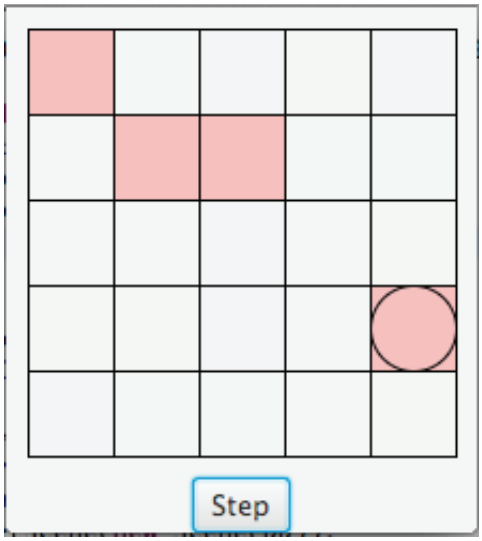
The following 4 pages show an example of how the robot's belief state changes as it proceeds to move in random directions in its colored grid world. The opacity of the grid colors are proportional to the entries in the robot's belief state (i.e. The more confident the robot is that is located at a particular position, the darker the color of the square.) The numerical belief state probabilities are displayed to the right of the figures. The transparent circle indicates the true location of the robot as it navigates through the grid.

This example is a simple one, as the robot only moves through the right column of the grid and sometimes stays in the same spot for multiple moves. However, due to its simplicity, I can easily make some observations that verify that the algorithm is working as expected. Here are several things to observe about the robot's inference:

The graphical user interface I am using is based in the appendix.

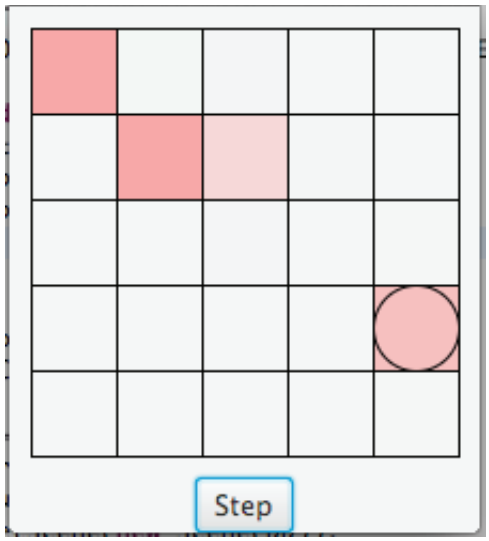


Grid Map



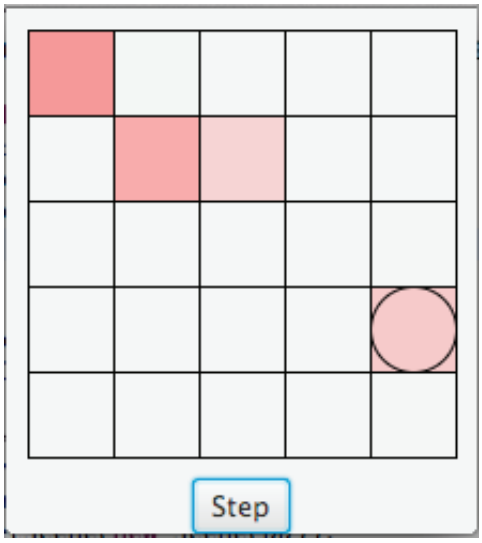
Move 1
Color read: R

.2136 .0097 .0097 .0097 .0097
.0000 .2136 .2136 .0097 .0097
.0000 .0000 .0097 .0000 .0097
.0097 .0097 .0097 .0000 .2136
.0097 .0097 .0097 .0000 .0097



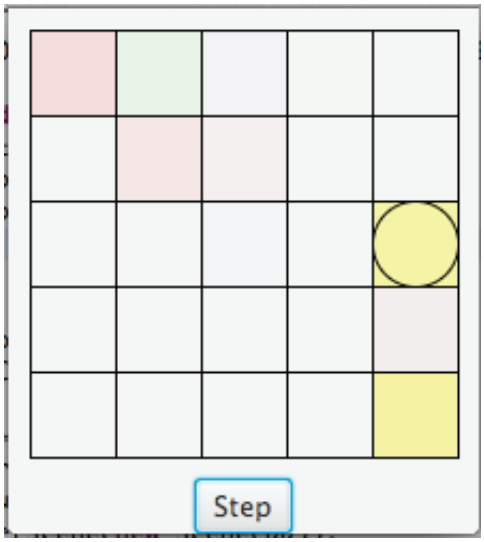
Move 2
Color read: R

.3125 .0098 .0053 .0008 .0008
.0000 .3125 .1166 .0053 .0008
.0000 .0000 .0053 .0000 .0053
.0008 .0008 .0008 .0000 .2145
.0008 .0008 .0008 .0000 .0053



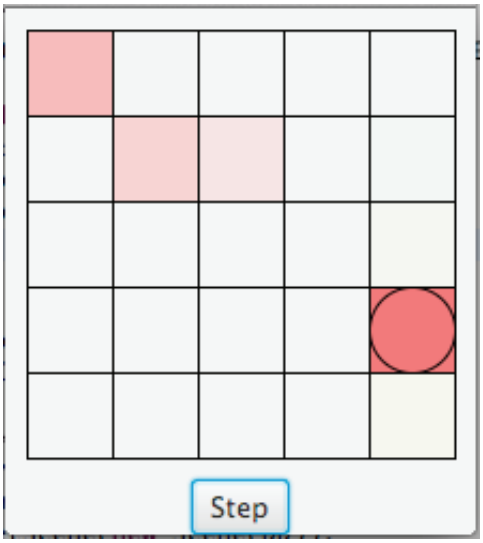
Move 3
Color read: R

.3735 .0115 .0024 .0002 .0001
.0000 .2962 .1295 .0022 .0002
.0000 .0000 .0023 .0000 .0041
.0001 .0001 .0001 .0000 .1734
.0001 .0001 .0001 .0000 .0041



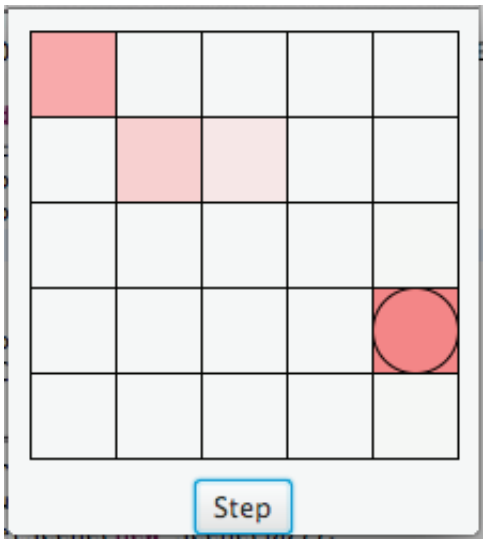
Move 4
Color read: Y

.0957 .0578 .0121 .0091 .0000
.0000 .0620 .0256 .0112 .0006
.0000 .0000 .0113 .0000 .3379
.0005 .0006 .0002 .0000 .0300
.0000 .0000 .0000 .0000 .3454



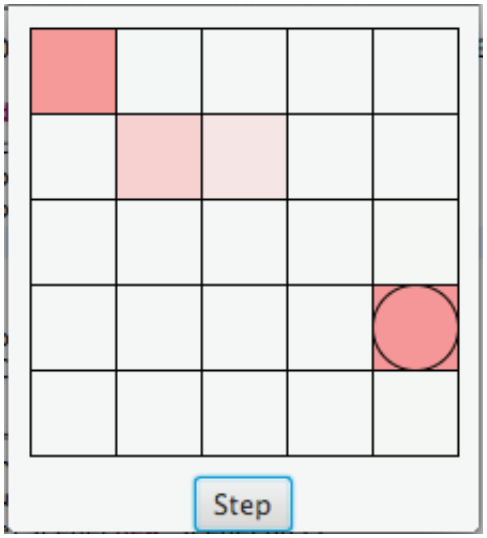
Move 5
Color read: R

.2282 .0068 .0031 .0010 .0003
.0000 .1373 .0640 .0014 .0105
.0000 .0000 .0015 .0000 .0213
.0000 .0000 .0004 .0000 .4920
.0000 .0000 .0000 .0000 .0321



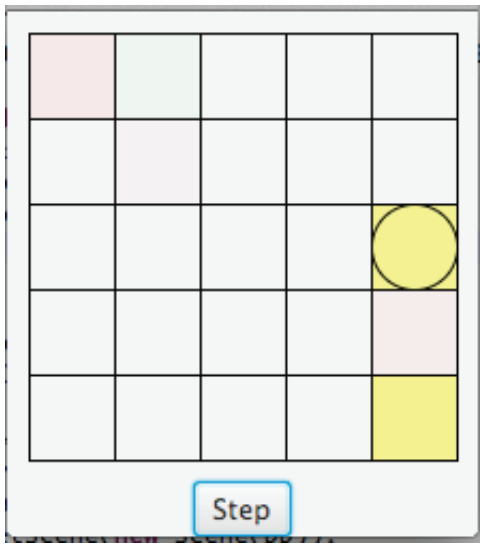
Move 6
Color read: R

.3009 .0074 .0015 .0001 .0002
.0000 .1503 .0623 .0015 .0007
.0000 .0000 .0013 .0000 .0108
.0000 .0000 .0000 .0000 .4513
.0000 .0000 .0000 .0000 .0116



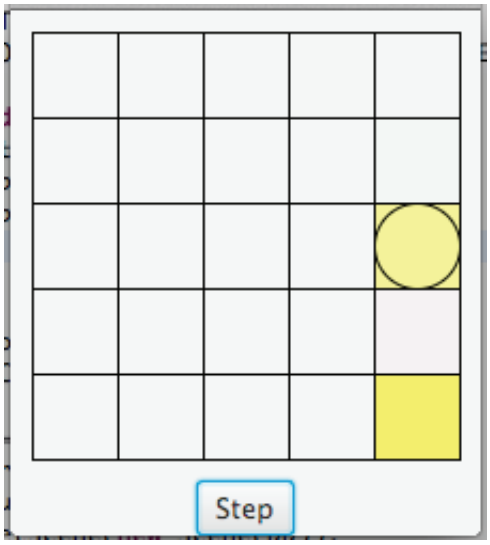
Move 7
Color read: R

.3738 .0086 .0013 .0001 .0000
.0000 .1521 .0635 .0012 .0002
.0000 .0000 .0012 .0000 .0088
.0000 .0000 .0000 .0000 .3800
.0000 .0000 .0000 .0000 .0091



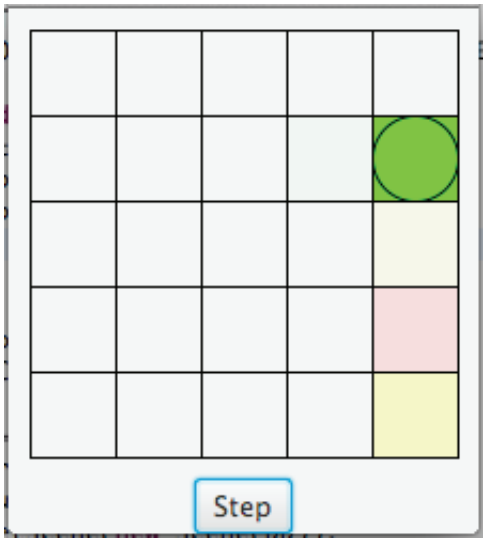
Move 8
Color read: Y

.0539 .0256 .0035 .0028 .0000
.0000 .0180 .0074 .0031 .0005
.0000 .0000 .0031 .0000 .4176
.0000 .0000 .0001 .0000 .0371
.0000 .0000 .0000 .0000 .4273



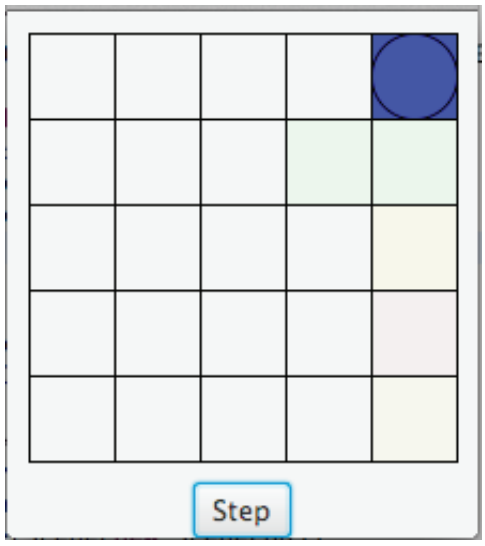
Move 9
Color read: Y

.0002 .0001 .0001 .0020 .0002
.0000 .0001 .0000 .0002 .0068
.0000 .0000 .0000 .0000 .3027
.0000 .0000 .0000 .0000 .0173
.0000 .0000 .0000 .0000 .6700



Move 10
Color read: G

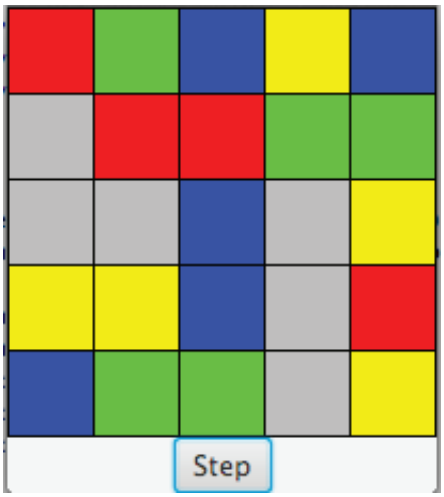
.0001 .0012 .0002 .0002 .0009
.0000 .0000 .0000 .0186 .6365
.0000 .0000 .0000 .0000 .0588
.0000 .0000 .0000 .0000 .0940
.0000 .0001 .0000 .0000 .1893



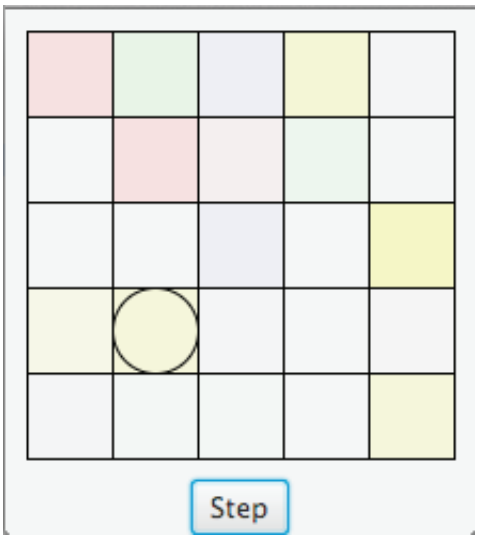
Move 11
Color read: B

.0001 .0001 .0022 .0011 .8050
.0000 .0001 .0011 .0376 .0410
.0000 .0000 .0001 .0000 .0486
.0000 .0000 .0001 .0000 .0250
.0001 .0000 .0000 .0000 .0379

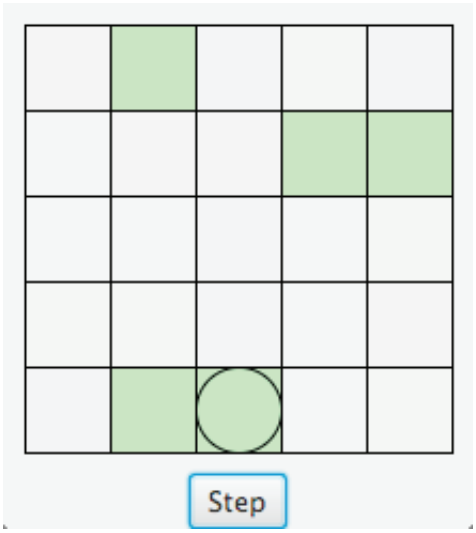
Map



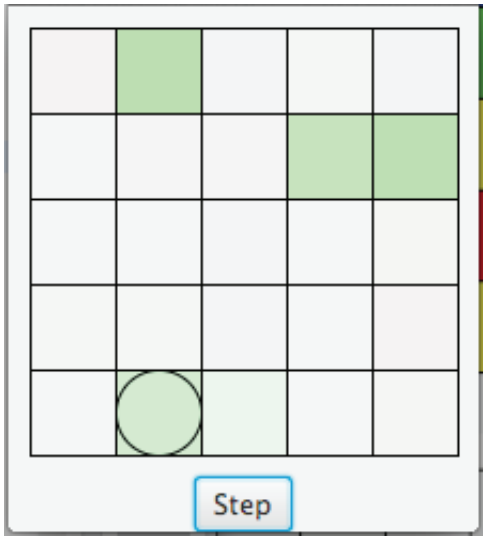
M3
Y



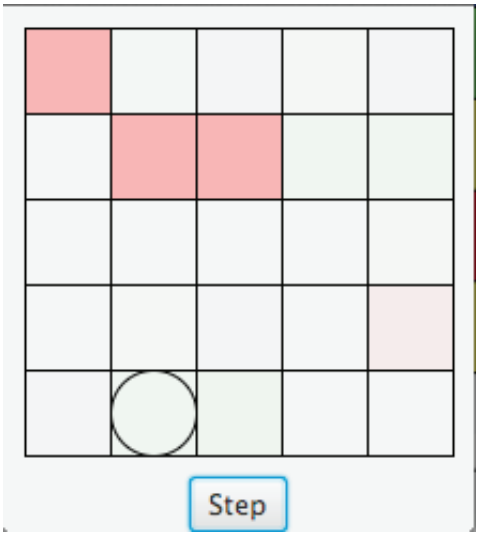
M1
G



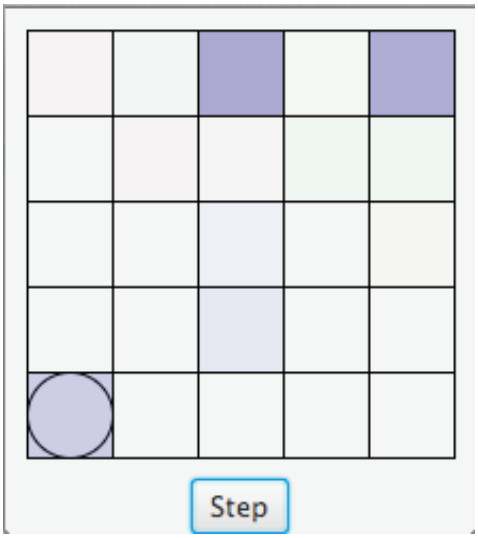
M4
G



M2
R



M5
B



7 References

Stuart Russell and Peter Norvig. "Artificial Intelligence: A Modern Approach" 3 ed. 2010.

8 Appendix