CSE 368 - AI: Homework 6 Markov Chains & Applications

In this homework you will implement algorithms that are built around the transition model of Markov Chains, which are a special case of a Bayes Nets that happen to be extremely useful in practice.

1) **(30) Markov Weather** In the `markov.py` file there is class called `weatherModel`. It has a member for a 3×3 transition matrix which you should train on the supplied data. The handout contains a small artificial dataset, and an extra file that has the actual weather data from Buffalo from last year.

- Convert the "raw" data into the right sequence to you can train a 3×3 matrix as discussed in class. Use the order `['sun', 'clouds', 'rain']`.
- Use the resulting transition model to predict the following:
  - The probability distribution over possible weather states given that it we have `'sun'` today.
  - Roughly, how many days into the future you think this prediction model will be useful.

Store your answers as `pTomorrow` and `predictionHorizon` as part of the class. This will make it easier for us to grade rather hand having an additional written homework question.

2) **(10) (Written) Draw State Diagram** Draw the state transition diagram for the weather model and label the transitions according to probabilities computed from the supplied data. There should be three states, one corresponding to each possible state of the weather.

3) **(15) Random Robot Walk** The robot will use a random motion model (motion can be any of the possible transitions) in order to build a Bayes filter for localization. Use the helper functions in the maze class to build a function that constructs the state transition matrix for random motion (not moving should be an allowable action). The function `ARandomWalk(self)` should be part of the robot class and return a matrix sized appropriately for the maze on which the robot was initialized, see `__init__()` in the robot class. The code provides a `step()` function for the robot. After initializing `A` you should be able to use it to predict possible future states of the robot. The matrix is also used by the `randomize()` function to get the steady-state location distribution of the robot, i.e. its location distribution after a long time wandering around.

4) **(15) Observations** Implement the function `obsLiklihood(self,o)` that takes an observation that is a list of length 4 and has entries zero or one. Each 'one' represents the likely presence of a wall or maze edge, and 'zero' indicates free space. The order is $[up, left, down, right]$, i.e. counter clock-wise starting from up. Each wall sensor is not perfectly reliable and returns the wrong result `obsError=0.2` of the time. Each direction is sensed independently. The function should return a vector that is the length of the state space where each entry is the likelihood of the state having generated the particular observation `o`.

5) **(30) Bayes Filter** This is a simplified version of the algorithm that underpins MANY estimation and robotics problems, from mapping to smoothing out noisy sensors in airplanes. It takes your forward prediction over states

from the transition matrix and improves the guess by adding information you get from sensors. Implement a such a filter for the robot using the two functions from the previous problems. You could think of this as a added module that is attached to a robot and tries to figure out the current position. The uniformly random motion, only means that all possible transitions are considered equally.

Use the two observation sequences `obs1` and `obs2` to answer the following questions and put the into the appropriate class files. These answers will be worth 10 of the points.

- What are the most likely locations after observing sequence `obs1` and `obs2`? Please store your answers in `loc1` and `loc2` respectively.
- In each sequence please store a list of items in which the observation has returned a faulty sensor value (or was kidnapped). (you don't need to do this automatically, but can inspect the Bayes filter). Store your answers in `errors1` and `errors2` respectively. For example, observation 6 `[1,1,0,1]` in `obsA` is obviously such an observation since there is no `[1,1,0,1]` wall configuration in the maze! Bu inspecting the Bayes filter, you should also be able to figure out which one of the sensors is probably malfunctioning. To aid your inspection, we provided some visualization functions `pList, pShow, robotShow`. Note that they rescale probability to be more visually helpful, if you want the raw probability values you should access them directly from `robot.prob`.

(Extra Credit) **(10) Kidnapping alarm!!** Add a field to the robot class called `kidnapped` that indicates the robot has been randomly taken and re-located. It should be set automatically as part of updating the `bayesFilter`. In addition to kidnapping, what are the other situations where the alarm might be triggered?

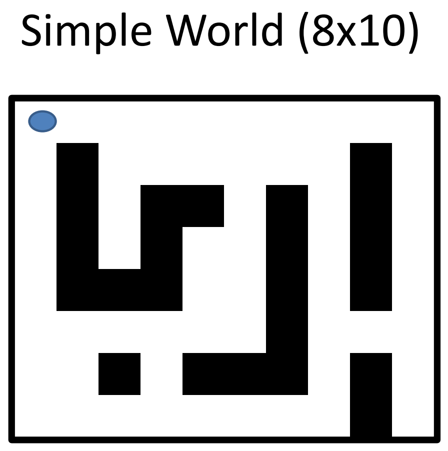

Figure 1: Simple maze world similar to the one used in the `markov.py`. The maze class does a few useful things like checking for neighbors, returning observations, and going between linear states (there are 80 of them) to [x,y] maze coordinates.