CSE368: Assignment 3

Introduction

In this assignment, you will be implementing both value iteration and policy iteration to solve robot maze world problem. There are 2 functions you need to implement: `valIter()` and `polIter()`, which are in `mdp.py` file. After implementation, submit your `mdp.py` on Autograder. The other file, `markov.py` has some helper functions for robots to present, move, and etc., which will be needed for robot maze.

Functions to implement

`valIter():`
Here you'll implement Value Iteration. You'll need to define the value of five possible actions at each state. You'll call stateReward, stateSize, gamma and the correspondent transition matrices. Use these functions to calculate the utility value iteratively for each state. Find and return the maximum value array, which includes all the maximum utility values for all the states. Make sure to update self.value and return this variable.

`polIter():`
Here you'll implement Policy Iteration. Similar to `valIter()`, you'll need to define the value of five possible actions at each state. You'll call stateReward, stateSize, gamma and the correspondent transition matrices. Use these functions to calculate the utility value for each state. Instead of finding the maximum utility value array, you'll find the argument array that gives the maximum utility values. Return the argmax of the utility values array which includes all the actions that created the largest utility value for each state. To create the policy array, you'll have to call actions, which defined at the beginning of `mdp.py`. Here you'll convert the numbers to words and return policy in an array of words. Note that you need to update self.policy variable and return this.

Hint

It is helpful to use numpy library to find maximum values.

Helper Functions

1. **In `markov.py`:**

   - `maze class:`

- __init__(self,world), which is initializer for the maze. It takes world as an input, which is initializing the information of the board we are working on.
- state2coord(self,s), which is the function transferring state to grid world coordinate (x,y), It takes states s as an input, outputs the coordinates of the states.
- coord2state(self,c), which is the inverse of state2coord(self,s). It transfers grid world coordinates (x,y) to states. It takes coordinates c as an input, outputs the states.
- numNbrs(self,s), which converts the states to numbers (0-79). It takes states s as an input and returns integers.
- nbrList(self,s), which returns neighbors index of a given state (0-79). It takes states as input and returns index of neighbors.
- actionList(self,s), which converts index of neighbors informs of actions. The 4 actions are ['U', 'D', 'L', 'R']
- observation(self,s), which returns the list of observations at state s. It returns [up, left, down, right]

❖ `robot class:` Includes the maze model, a random action model. It estimates over the possible robot location in the maze. Bayes filter for localization is implemented in this class. You can think of it as trying to figure out the robot location from a stream of sensor measurements of the form [0,1,1,0] where the order is [up,left,down,right] and zero indicates free space and 1 indicates a maze edge or a wall.

- ARandomWalk(self), which returns an initialized 80*80 random walk zero matrix.
- __init__(self,maze), which initializes some variables. It takes maze as an input. Defined transition matrix A as ARandomWalk.
- mpower(self,A,n), which takes random walk A and a power n as inputs, returns the matrix A raise to the power of n..
- randomize(self), which gets initial condition after long wandering. It multiplies the transition matrix A by itself 1000 times. Use the first column of the result to present the probability of the nodes walked to the state 0.
- step(self), this is how A walks.

2. In `mdp.py`:

❖ `MDPMaze class:`

- __init__(self, maze, stateReward), which initializes variables and place holders. It takes maze class from markov.py, stateReward as inputs.
- ARandomWalk(self), which creates random walk matrix A.
- computeTransitionMatrices(self), which creates transition matrices for Aup, Aleft, Adown, Aright and Astop.