TCSS 435
Programming Assignment 3

Objective: Expressing the problem as a Markov Model and implementing the Markov Chain transitions.

Instructions: In this assignment you will implement the Markov chain to see the progress an agent makes over time. Theoretically solving the uncertainty problem includes progressive construction of a solution:

- Establish the problem's components
    - Start State (Initial/Prior probabilities)
    - Transition Table (Transition probabilities)
- Establish agent transitions using Markov Model

Explanation: To understanding the setup and capture agent transitions through time, consider a simple 3 x 3 maze as shown in Figure 1.

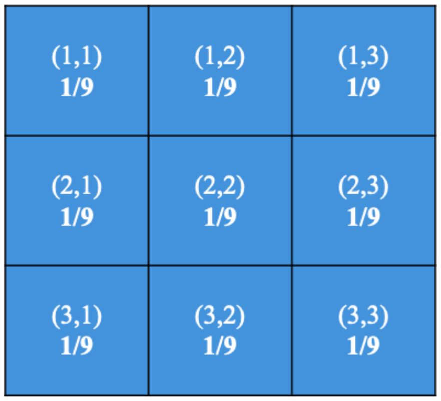
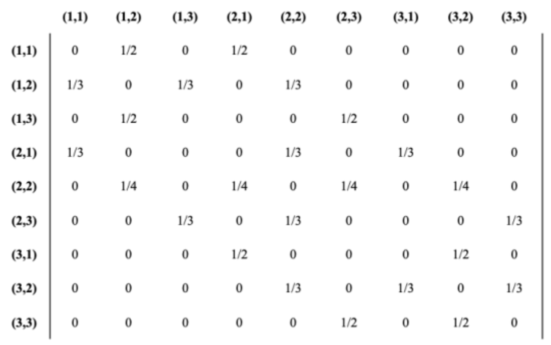


Figure 1: A 3 x 3 maze with cell number (row, column) and prior probabilities

Our agent is a Markov agent, i.e. the agent moves randomly from cell to cell, with the probability distribution of the next cell depending only on the current cell, not on the history of how it got to the current cell (this is the Markov property). Moreover, we assume that the agent is equally likely to start in any cell of the 3 x 3 maze (i.e. prior probability of each cell is 1/9).

We now model the movement of the agent as a Markov chain. The state of the Markov chain is the cell occupied by the agent. We let the time index *n* refer to the n^{th} cell visited by the agent. So, we make a discrete-time Markov chain. Specifically, we let X_n be the state (cell) occupied by the agent on step (or time or transition) *n*. The initial cell is X_0. The cell after the first transition is X_1, and so forth. Then $\{X_n: n \geq 0\}$ is the discrete-time Markov chain; it is a discrete-time discrete-state stochastic process. The agent is in cell Xn (a random variable) after making *n* moves, after having started in cell X_0, which could also be a random variable. We specify the evolution of the Markov chain by specifying the one-step transition probabilities. We specify these transition probabilities by specifying the transition matrix (see Figure 2).



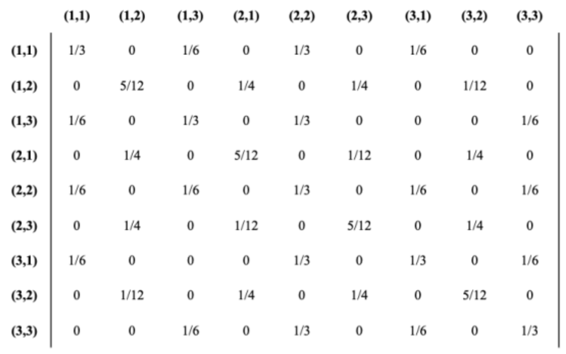
$$P = \begin{array}{c|ccccccccc|}
 & (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \\
(1,1) & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\
(1,2) & 1/3 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 \\
(1,3) & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\
(2,1) & 1/3 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 \\
(2,2) & 0 & 1/4 & 0 & 1/4 & 0 & 1/4 & 0 & 1/4 & 0 \\
(2,3) & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 1/3 \\
(3,1) & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 \\
(3,2) & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\
(3,3) & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0
\end{array}$$

Figure 2: Transition Matrix

Let’s try to solve the following problem: What is the probability of the agent starting in cell (1,2) and reaching cell (1,2) again in two transitions (2-steps)?
Solution: There are multiple paths

```
    i.   (1,2) > (1,1) > (1,2) = 1/3 * 1/2 = 1/6
   ii.   (1,2) > (2,2) > (1,2) = 1/3 * 1/4 = 1/12
  iii.   (1,2) > (1,3) > (1,2) = 1/3 * 1/2 = 1/6
```

Summing all the individual path probabilities, we get `1/6 + 1/12 + 1/6 = 5/12`
Hence, the probability of the agent starting in cell (1,2) and reaching cell (1,2) in 2-steps = `5/12`
Another way to arrive at the solution for the above question is matrix multiplication. If we raise the transition matrix to the power 2 (P^2), we will get the transition matrix or the probability for the agent to reach any cell starting from any other cell in two transitions (2-steps). Below is the P^2 matrix:



$$\begin{array}{c|ccccccccc|}
 & (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \\
(1,1) & 1/3 & 0 & 1/6 & 0 & 1/3 & 0 & 1/6 & 0 & 0 \\
(1,2) & 0 & 5/12 & 0 & 1/4 & 0 & 1/4 & 0 & 1/12 & 0 \\
(1,3) & 1/6 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 1/6 \\
(2,1) & 0 & 1/4 & 0 & 5/12 & 0 & 1/12 & 0 & 1/4 & 0 \\
(2,2) & 1/6 & 0 & 1/6 & 0 & 1/3 & 0 & 1/6 & 0 & 1/6 \\
(2,3) & 0 & 1/4 & 0 & 1/12 & 0 & 5/12 & 0 & 1/4 & 0 \\
(3,1) & 1/6 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/6 \\
(3,2) & 0 & 1/12 & 0 & 1/4 & 0 & 1/4 & 0 & 5/12 & 0 \\
(3,3) & 0 & 0 & 1/6 & 0 & 1/3 & 0 & 1/6 & 0 & 1/3
\end{array}$$

Figure 3: Transition Matrix after 2-steps

Using the above transition matrix multiplication, we can answer other questions like: If the agent starts in cell (1,1), will it reach the goal state (2,3) in 2 -steps? Answer is **no** as probability of cell (1,1) > (2,3) = 0 after 2-steps.

Let’s discuss the implementation details:
Step 1: Defining our Maze
Update your `MazeRunner.py` source code (from Assignment 0) to create a Maze using following configurations *(Note: The `MazeRunner.py` will be our tester file)*:

- Maze configuration (M x N): We will test for following two Maze sizes- default size (10x10) and size of your own choosing.

- You can choose the theme to be light or dark and pattern to be vertical or horizontal.
- Goal State will be generated randomly using random number generator.
- Maze 1: Set loop percent to be 50 and,
- Maze 2: Set loop percent to be 0.

Step 2: Capturing Transition matrix and implementing Markov transitions

Create a class file `Markov.py` to:

- Represent the transition matrix: You will need the maze map to determine the directions in which an agent can move. If the agent can move in:
  - 1 direction, then transition probability is 1
  - 2 directions, then transition probability is 1/2
  - 3 directions, then transition probability is 1/3
  - All 4 directions, then transition probability is 1/4
- You will implement the Markov transitions to address the following questions:
  - i. Problem 1: Starting at a random location, where will the agent most likely be (cell position) in n = 100 steps
    - Print transition matrix after every 10 steps.
  - ii. Problem 2: Starting at a random location, after how many steps does the agent reach the goal state?
    - Print the n – step in which the agent will reach the goal state.
    - Print transition matrix when agent reaches the goal state.
    - Show agent footsteps across each path (at-most 3 paths) from start to goal.
  - iii. Extra (Optional): What will be the steady state distribution (i.e. distribution at n = infinity)?
    - Print the steady state transition matrix.

Notes:

- *Use `NumPy` to perform matrix multiplication (function to use is `matmul`).*
- *You cannot jump over the wall.*
- *Implementation-wise you don't need to solve these problems individually, i.e. while performing n = 100 iterations, if you reach the goal state then report solution to problem 2 and vice-versa.*

Step 3: Passing inputs

Your program *`MazeRunner.py`* will accept input arguments from the command line in the following format: `MazeRunner.py [size][loopperc]`

- `[size]` maze size options are `10` (10 x 10) or `number of my choosing (m x m)`
- `[loopperc]` loop percent options are 0 or 50.

Step 4: Generating Outputs

Your program will generate 2 different outputs – GUI & and a readme file.

1. GUI Output: Trace agent's path(s) from its start state to the goal state based on the states (cell) that are reachable. Trace 1 (at-least) to 3 (at-most) paths to the goal state from start state.

- Example: Consider transition matrix from Figure 2 and Figure 3. If the agent starts in cell (1,2) and goal cell is (2,1) then we have the following path (we get only 1 path) after 2 steps: `[(1,2),(2,2),(2,1)]`

Note: You might have noticed we did not initialize an agent in step 1. Here is how you can use agents to trace multiple paths:

```
a = maze.agent(m,1,1,footprints=True, filled=True, color=maze.COLOR.red)
b = maze.agent(m,1,1,footprints=True, filled=True, color=maze.COLOR.green)
c = maze.agent(m,1,1,footprints=True, filled=True, color=maze.COLOR.yellow)

path1 = [(1,2),(1,3),(1,4),(1,5),(2,5),(3,5),(4,5)]
path2 = [(2,1),(3,1),(4,1)]
path3 = [(2,1),(2,2),(3,2)]

m.tracePath({a:path1, b:path3, c:path2})
```

2. `Readme.txt:` In your readme file you will include the following information:
   For each maze `size` report:
   - Report the transition matrix after every 10 steps.
   - Report the n (i.e. after how many steps) required by the agent to reach the goal state.
   - Report transition matrix when agent reaches the goal state.
   - Extra (Optional): Report the steady state transition matrix.

Submission Guideline: Zip & upload the following files to the assignment submission link:

- `Markov.py`: Class file that represents the transition matrix and implements Markov transitions.
- *`MazeRunner.py`*: The main source code file that accepts the input and connects your pyamaze to the Markov model.
- `Readme.txt`: Output for each maze size as explained in the output section.