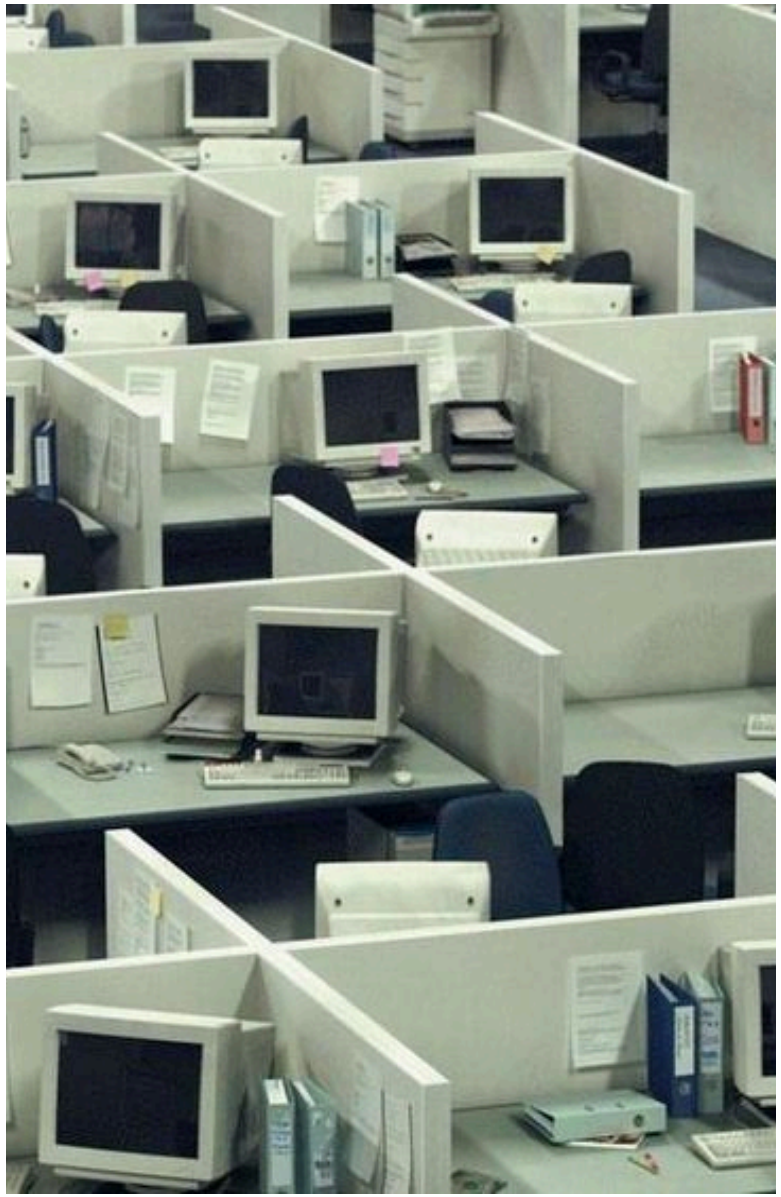


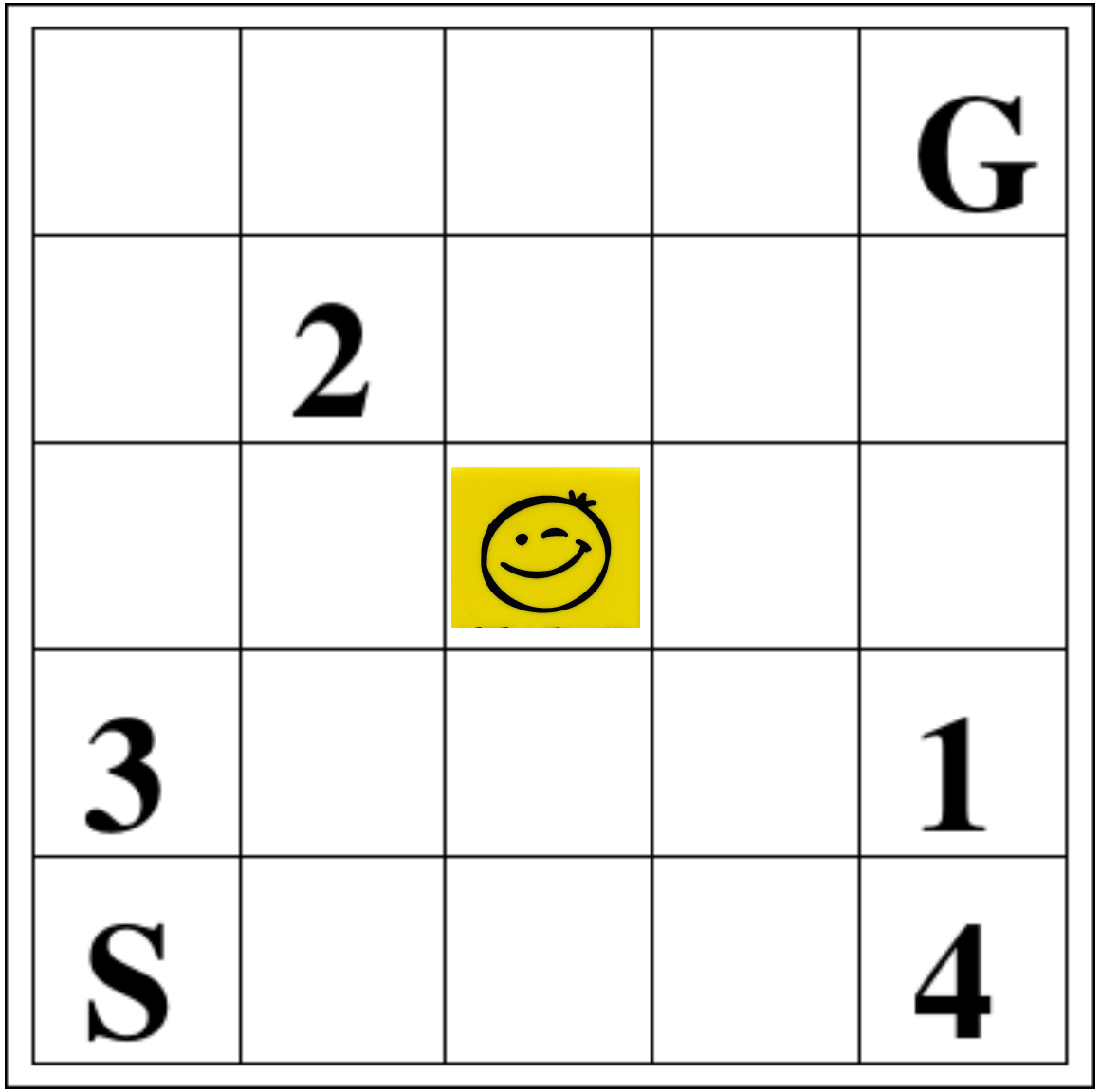
A short story to help develop an intuition about POMDP

Imagine yourself (agent) as a tiny person in the middle of huge IT corporate workspace. You are trying to get to the exit (goal) as soon as possible (reward). The workspace is a huge grid of cubicles (state space). When you start exploring, you move into different cubicles. You are able to observe different things within cubicles such as desks, workstations, chairs etc and maybe a washroom or a coffee machine near a few cubicles but not the cubicle number. Based on these observations (partially observable data), you may have an idea about where you might be (belief) but you don't exactly know the cubicle number(state) in the workspace (POMDP). Now, in order to get help in navigating the workspace, you call up a friend who works in that office. She tells you " if you are in cubicle no x (current state) take a right and walk straight to fins the exit"(policy). However, this suggestion (policy) is of no use to you as you do not know the cubicle number (state) you are in (complexity in solving POMDPs)

Had it been a scenario in which the cubicles are numbered and you are able to observe the cubicle number (state) along with other observations directly (fully observable MDP), you would be easily able to navigate the workspace with the help of your friend's suggestions (policy) based on your current position.

POMPD

Aishwarya Pothula



GOAL

The project aims to implement a POMDP GridWorld environment and a POMDP learner to navigate the GridWorld environment

Problem Setup

- The environment is a 15 x 25 grid. The environment is deterministic. The start position is [0,0,0] and the goal location is [14,24]. The first two indexes of state specify the grid index and the third index in the position specifies the orientation of the agent
- The agent can perform “Forward”, “Backward”, ”Right Turn” and “Left Turn”. The grid location of the agent usually changes when the first two actions are performed. Case one of exceptions are when the agent tries to go out of the grid or when it encounters an obstacle. Case two is that there are 20% and 10% probabilities respectively for ‘moving’ and ‘turning’ actions that they are not performed by the agent; the agent retains its position and orientation within the grid. When the ‘turn’ actions are performed, the agent just changes its orientation and not the grid location
- The environment has 1500 distinct states ranging from 0 to 1499. We derive this number by assigning observations to states. The size of the observation space is 15 x 25 x 4. While 15 and 25 represent the size of the grid, the 4 represents the four orientations of the agent.
- The environment gives a reward of 100 when the agent reaches the goal position, -100 when the agent encounters an obstacle or tries to move out the grid. For all other actions, the agent is given a reward of 0.

What is POMDP

HMM

Implies that the Markov Model underlying the data is unknown to the agent. Only observable data and not information about the state is available to the agent

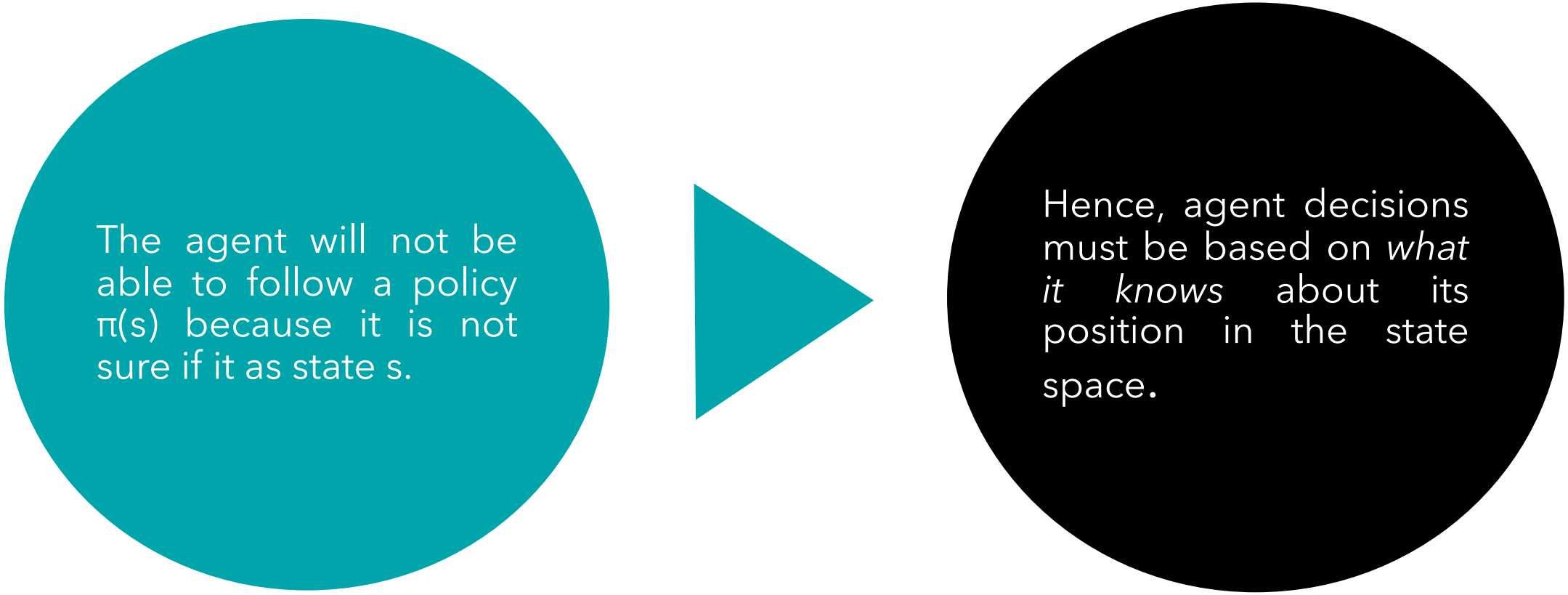
POMDP

A Partially observable MDP or POMDP is somewhere between a Hidden Markov model and a fully observable MDP. State information is only partially available

MDP

Fully observable ie agent has full information about the state it is in in the state space. Also, the Markov assumption for $P(s'|s,a)$ holds which implies that the current policy π^* depends only on the current state

Problem with Env being only partially observable



Note: Policy π(s) is a decision strategy which tells what action is to be taken given that an agent is in state s

Belief States

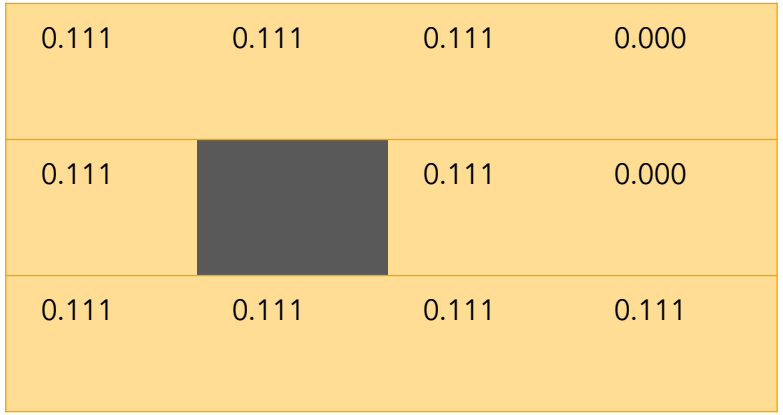
Though a POMDP cannot tell where it is in the state space, it forms a beliefs about its position in the state space

Beliefs are probability distribution over states. A belief b(s) is the probability associated by belief b to the agent being in state s

Initial belief state consists of equal probabilities assigned to all non-terminal states and states other than the ones containing obstacles.

For example, for an environment with obstacles in [0,3] and [1,3] and goal state in [1,1], the initial belief state would be

<1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,0,0>



POMDP Representation

A POMDP is fully specified by

- A Reward Function R(s)
- A Transition Model P(s'|s,a)
- An Observation Model P(o|s,a)

A belief state is updated using the following equation

$$\mathbf{b}'(\boldsymbol{s}) = \boldsymbol{P}(\boldsymbol{o}|\boldsymbol{s}', \boldsymbol{a}) \sum_{\boldsymbol{s}} \boldsymbol{P}(\boldsymbol{s}|\boldsymbol{a})\, \boldsymbol{b}(\boldsymbol{s})$$

```
def update_belief_states(self, s, nn_state, action):
        # state belief update
        b = self.prob * self.sigma_transition(nn_state ,action)
```

```
def sigma_transition(self, s,action):
        sigma = 0
        for i in (-1, 0, 1):
            for j in (-1, 0, 1):
                for k in (0, 1, 2, 3):
                    state_index = [s[0] + i, s[1] + j, k]
                    prob = self.transition_prob(s, state_index, action)
                    if  prob != 0.0:
                        sigma += (prob * self.belief_hash.get((state_index[0], state_index[1], state_index[2]),0))
```

What's the solution – how to find optimal policy

POMDPs can be solved by

- Converting POMDP to MDP and applying Value Iteration
- Generalizing Value Iteration to work with POMDPs
- Performing look ahead and then applying Value Iteration
- Point based Value Iteration

***For* our project we are using Look ahead .**

1 Try all actions from a state to form a tree of the look ahead values. Every branch represents a sequence of actions and observations available to agent and corresponding beliefs.

```
def get_states(self, state, nn_state, action):
        array = []
        if state != nn_state:
            array.append(state)
        for act in list(range(self.actions)):
            s = self.go_to_next_state(state, act)
            if s != nn_state and s not in array:
                array.append(s)
        for ix in list(range(len(array))):
            for j in range(4):
                s = self.go_to_next_state(array[ix], j)
                if  s != nn_state and s not in array:
                    array.append(s)
        for ix in list(range(self.actions)):
            s = self.go_to_next_state(nn_state, ix)
            if s != nn_state and s not in array:
                array.append(s)
```

2 Perform backup of the utility values along the branches from the leaf. The leaf at the end of each branch corresponds to the belief state reachable via that sequence of actions and observations

```
self.q_table[current_state[0]][current_state[1]][current_state[2]][action] = act +
                    (reward + self.gamma * max(nsa) - q_state) * oh_belief * self.alpha
```

3 Pick the branch with the highest expected value

```
def take_action(self, state):
        random_number = random.random()
        if random_number < self.epsilon: return random.randint(0, actions - 1)
        else:
            s, q_table_max = self.q_table[state[0]][state[1]][state[2]][:], self.q_table[state[0]][state[1]][state[2]][:][0]

            for ix in list(range(actions)):
                if q_table_max <= s[ix]:
                    q_table_max = s[ix]

            index_max = [ix for ix in list(range(actions)) if s[ix] == q_table_max]
            pick = random.randint(0, len(index_max) - 1)
            return index_max[pick]
```

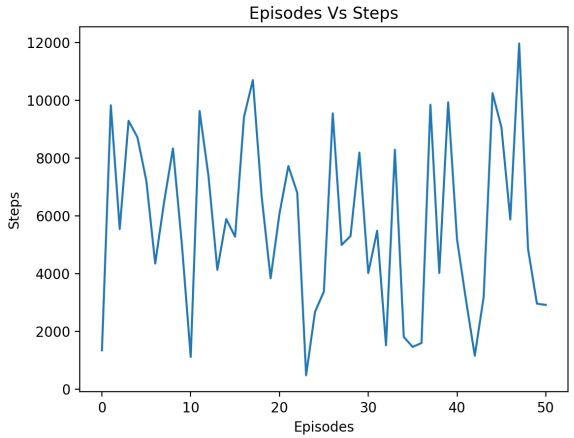
Execution and Result

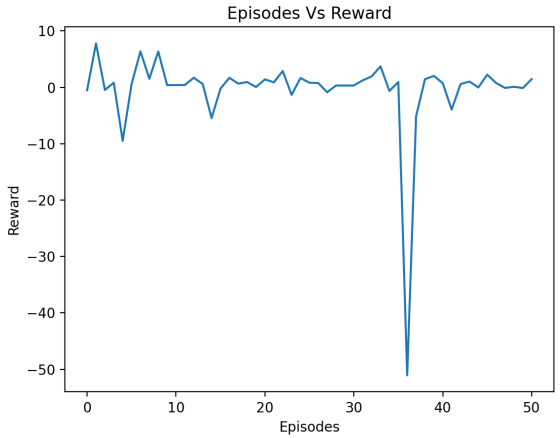
Execution

- To run the program, extract files from the submitted zip folder
- Run $ python3 pompdp.py

Result

- The learner has been trained over 50 episodes . The number of steps taken at the 50th episode are 2958
- The average reward over the 50 episodes is 2.2012
- The following graphs have been generated from console logs of steps episodes and rewards





Sources

- http://cs.brown.edu/research/ai/pomdp/tutorial/
- https://www.youtube.com/watch?v=9G_KevA8DFY&t=768s
- https://github.com/mohsen-imani/POMDP-Learning
- https://www.cs.ubc.ca/~carenini/TEACHING/CPSC502-11/LECTURES/lecture13-2011.pdf