REPORT

Rohith Parjanya
(M20CS009)

I. CODING PROBLEM

Reinforcement learning:

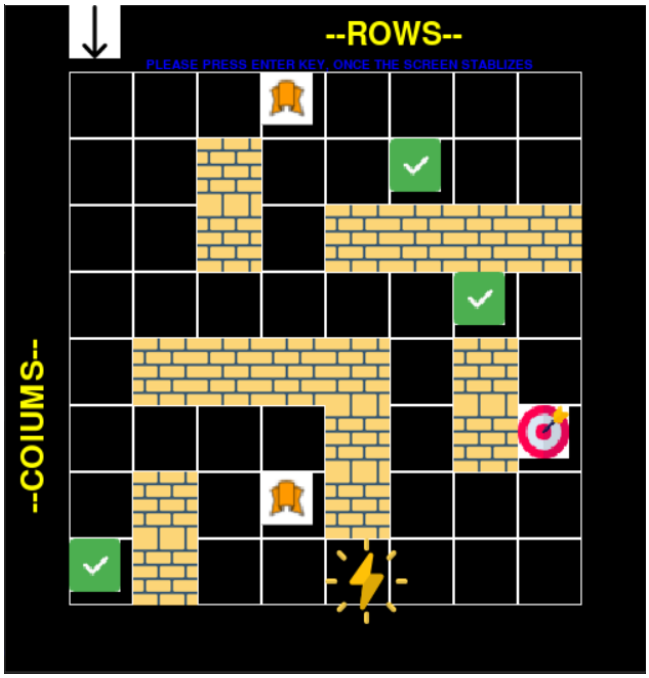
The approach used is Adaptive Dynamic programming and on top of it Value iteration is used to get the values that it quantifying how good is the state for the agent to be in and the action for which the maximum value is achieved from any given state that action is used to update the policy.

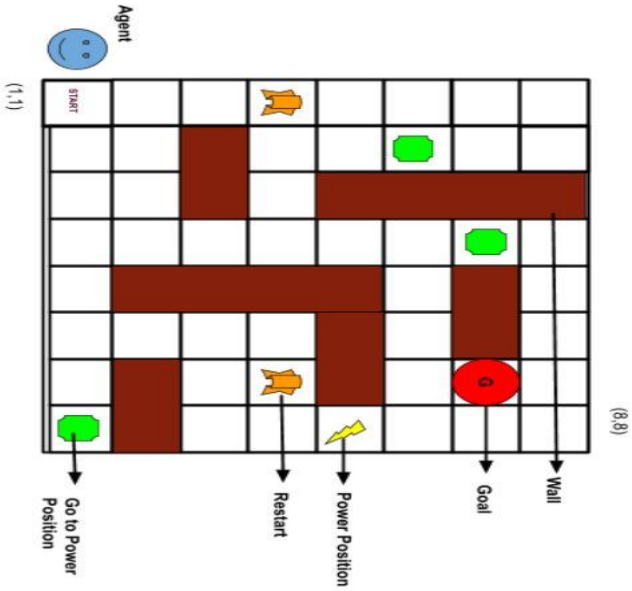
The environment constraints:

1. Go to Power Position : On Reaching here, agent will start its next move from “Power Position” as shown
2. Goal: The location of the goal should be randomly selected from the range (x>5, y>5) eg. (6,6), (8,8), (8,6) etc.,
3. Restart: The Agent will need to go restart and go to position (1,1)
4. Brown Blocks : Walls (Agent isn't allowed to penetrate through these positions)
5. Start: Agent will be starting from (1,1) every time the game is started

The setup:

The start state is represented by an arrow ‘→’





A. Theory

1. **Policy design:** The policy is defined as π , which is a 4 tuple structure (S, A, P, R)
   - **S** → States are nothing but the blocks in 8 * 8 grid shown above.
   - **A** →Actions of the agent include ('L'→Left, 'R'→Right, 'U'→ Up, 'D'→ Down)
   - **P** → Probability of transition, But in ADP we agent itself must explore the transition PROBABILITY and the NOISE. But in our case there are no sensors involved it is obvious that the transition probabilities are given explicitly in program. Here Gamma (γ) is taken as 0.9 (that means 90% of the times agent moves in the correct direction) and NOISE = 0.1, which means 10% of the times agent deviates from the correct direction.
   - **R** →R(s) comprises of the reward function for the agent. It is significantly a real number which is awarded to the agent on reaching that state.

2. **Reward function design:**

   - The reward function here is defined as the agent reaches every empty block in the grid it receives '0' reward. R(Empty block) = 0
   - If the agent reaches any of the "RESTART STATES", then the agent receives a penalty. So, the reward at restart states is negative. R((4,1) and (4,7)) = -80
   - If the agent reaches any of the "GREEN POSITIONS," then the agent gets a slightly lower reward when compared to the GOAL state because the power position is always close to the GOAL states. So, R((5,8), (6,2), (7,3)) = +50
   - If the agent reaches the power position, it is clear that it is close to the GOAL state. So, the reward at power position is given approximately close to the GOAL state that is R(5,8) = +80
   - Finally the reward of the GOAL state is R(randomly chosen from ((6, 5), (6, 6), (7, 7), (7, 8), (6, 7), (8, 5), (8, 6), (8, 7),(8,8),(6,8))) = +100

3. **Knowledge Base:**

   - Agent while exploring primarily stores
     1. The values function outcomes (Utility) at any stage, which is nothing but how good it is for the agent to be in a particular state.
     2. The policy is the action with the highest possible value.
   - Here for both purposes, a HASH MAP/ DICTIONARY is being used for storing the values. For keeping the values dictionary, 'V' is used in the assignment. For storing the policy dictionary, 'Policy' is used

B. Code

1.Policy design:
At the initial stages a random policy is created :

```
# Define an initial policy
policy = {}
for s in actions.keys():
    policy[s] = np.random.choice(actions[s])
```

Then after exploration at every iteration the maximum action to be carried out is being stored in the policy dictionary.

Reward function:

```
# Define rewards for all states
rewards = {}
for i in all_states:
    if i == (4, 1):
        rewards[i] = -80
    elif i == (4, 7):
        rewards[i] = -80
    # power position
    elif i == (5, 8):
        rewards[i] = 80
    # green positions
    elif i == (1, 8):
        rewards[i] = 50
    elif i == (6, 2):
        rewards[i] = 50
    elif i == (7, 3):
        rewards[i] =50
    elif i == (GOAL[0], GOAL[1]):
        rewards[i] = 100
    else:
        rewards[i] = 0
```

Search algorithm:
My intuition writes the search algorithm. It includes considering local maxima from all possible one-step actions and repeating the same continuously.

2. Knowledge Base:

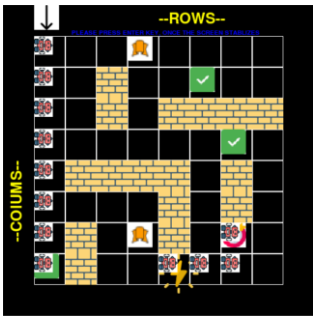
Policy dictionary at iteration 0: Includes all random actions.
{(1, 1): 'R', (1, 2): 'L', (1, 3): 'R', (1, 4): 'R', (1, 5): 'R', (1, 6): 'R', (1, 7): 'R', (1, 8): 'L', (2, 1): 'U', (2, 2): 'D', (2, 3): 'D', (2, 4): 'U', (2, 6): 'D', (3, 1): 'U', (3, 4): 'U', (3, 6): 'D', (3, 7): 'L', (3, 8): 'L', (4, 1): 'R', (4, 2): 'U', (4, 3): 'L', (4, 4): 'L', (4, 6): 'D', (4, 7): 'L', (4, 8): 'U', (5, 1): 'R', (5, 2): 'U', (5, 4): 'U', (5, 8): 'U', (6, 1): 'R', (6, 2): 'D', (6, 4): 'U', (6, 5): 'L', (6, 6): 'R', (6, 7): 'R', (6, 8): 'L', (7, 1): 'D', (7, 2): 'D', (7, 4): 'D', (7, 7): 'D', (7, 8): 'D', (8, 1): 'D', (8, 2): 'D', (8, 4): 'D', (8, 5): 'L', (8, 6): 'R', (8, 7): 'D', (8, 8): 'D'}

Value dictionary (V) at iteration 0:
{(1, 1): 0.0, (1, 2): 0.0, (1, 3): 0.0, (1, 4): 0.0, (1, 5): 0.0, (1, 6): 0.0, (1, 7): 0.405, (1, 8): 50.32805, (2, 1): 0.0, (2, 2): 0.0, (2, 3): 0.0, (2, 4): 0.0, (2, 6): 0.0, (3, 1): 0.0, (3, 4): 0.0, (3, 6): 0.0, (3, 7): 0.0, (3, 8): 0.0, (4, 1): 0.0, (4, 2): 0.0, (4, 3): 0.0, (4, 4): 0.0, (4, 6): 0.0, (4, 7): 0.0, (4, 8): 0.0, (5, 1): 0.0, (5, 2): 0.405, (5, 4): 0.0, (5, 8): 80.0, (6, 1): 0.405, (6, 2): 57.52805, (6, 4): 0.405, (6, 5): 0.32805000000000006, (6, 6): 0.2657205000000001, (6, 7): 0.81, (6, 8): 64.8, (7, 1): 0.32805000000000006, (7, 2): 46.5977205, (7, 4): 7.5280499999999995, (7, 7): 100.6561, (7, 8): 81.531441, (8, 1): 0.2657205000000001, (8, 2): 37.76806845, (8, 4): 6.0977205, (8, 5): 4.9391536050000004, (8, 6): 4.00071442005, (8, 7): 81.8915052978045, (8, 8): 73.66994898122165}

Agent Exploring States:
By clicking “SPACE BAR,” you can visualize the agent exploring states, and it is printed in the output section.

Path followed and Path cost:

By clicking the “ENTER KEY” after the screen stabilizes, the agent fully exploring the states, then the path is followed, and the path cost is printed on the screen.



Trial 1:

----------HASH MAP FOR STORAGE OF UTILITIES AT ITERATION-- 500
STATE: (1, 1) :: UTILITY : 127.7781451701158
STATE: (1, 2) :: UTILITY : 145.0953409385659
STATE: (1, 3) :: UTILITY : 163.93392234343762
STATE: (1, 4) :: UTILITY : 185.5494642317211
STATE: (1, 5) :: UTILITY : 209.1453492963896
STATE: (1, 6) :: UTILITY : 239.09092410770236
STATE: (1, 7) :: UTILITY : 270.0397028730918
STATE: (1, 8) :: UTILITY : 306.9077568242682
STATE: (2, 1) :: UTILITY : 122.27736843040262
STATE: (2, 2) :: UTILITY : 138.34624704304605
STATE: (2, 3) :: UTILITY : 159.26111767633537
STATE: (2, 4) :: UTILITY : 176.04817946086797
STATE: (2, 6) :: UTILITY : 227.43736880333
STATE: (3, 1) :: UTILITY : 115.57822603552113
STATE: (3, 4) :: UTILITY : 199.070759479171
STATE: (3, 6) :: UTILITY : 249.7558428332092
STATE: (3, 7) :: UTILITY : 287.1690180082766
STATE: (3, 8) :: UTILITY : 327.8442628757514
STATE: (4, 1) :: UTILITY : 129.21214240261918
STATE: (4, 2) :: UTILITY : 238.97841196839812
STATE: (4, 3) :: UTILITY : 213.93100517525724
STATE: (4, 4) :: UTILITY : 227.21160699636897
STATE: (4, 6) :: UTILITY : 223.2525888951373
STATE: (4, 7) :: UTILITY : 233.49908432031685
STATE: (4, 8) :: UTILITY : 364.5952950624026
STATE: (5, 1) :: UTILITY : 244.15911751891537
STATE: (5, 2) :: UTILITY : 270.94167050002704
STATE: (5, 4) :: UTILITY : 255.62779654422255
STATE: (5, 8) :: UTILITY : 423.3206271351833
STATE: (6, 1) :: UTILITY : 271.40793399957363
STATE: (6, 2) :: UTILITY : 307.9392829818212
STATE: (6, 4) :: UTILITY : 291.4522201839139
STATE: (6, 5) :: UTILITY : 329.2989561898043
STATE: (6, 6) :: UTILITY : 374.1235119912986
STATE: (6, 7) :: UTILITY : 426.08939453875445
STATE: (6, 8) :: UTILITY : 383.2386679132286
STATE: (7, 1) :: UTILITY : 244.26434988743034
STATE: (7, 2) :: UTILITY : 271.4146107051439
STATE: (7, 4) :: UTILITY : 274.6003551019701
STATE: (7, 7) :: UTILITY : 483.48785697955003
STATE: (7, 8) :: UTILITY : 426.1323651359705
STATE: (8, 1) :: UTILITY : 219.41147467048083
STATE: (8, 2) :: UTILITY : 239.59286739150983
STATE: (8, 4) :: UTILITY : 291.5527257672875
STATE: (8, 5) :: UTILITY : 329.29552298247637
STATE: (8, 6) :: UTILITY : 374.0983920823608
STATE: (8, 7) :: UTILITY : 425.294019440848
STATE: (8, 8) :: UTILITY : 383.4436775098124
----------HASH MAP FOR POLICY----------
STATE: (1, 1) :: ACTION : R
STATE: (1, 2) :: ACTION : R
STATE: (1, 3) :: ACTION : R
STATE: (1, 4) :: ACTION : R
STATE: (1, 5) :: ACTION : R
STATE: (1, 6) :: ACTION : R
STATE: (1, 7) :: ACTION : R
STATE: (1, 8) :: ACTION : L
STATE: (2, 1) :: ACTION : R
STATE: (2, 2) :: ACTION : R
STATE: (2, 3) :: ACTION : R
STATE: (2, 4) :: ACTION : U
STATE: (2, 6) :: ACTION : U
STATE: (3, 1) :: ACTION : U
STATE: (3, 4) :: ACTION : U
STATE: (3, 6) :: ACTION : R
STATE: (3, 7) :: ACTION : R
STATE: (3, 8) :: ACTION : U
STATE: (4, 1) :: ACTION : U
STATE: (4, 2) :: ACTION : U
STATE: (4, 3) :: ACTION : L
STATE: (4, 4) :: ACTION : U
STATE: (4, 6) :: ACTION : D
STATE: (4, 7) :: ACTION : R
STATE: (4, 8) :: ACTION : U
STATE: (5, 1) :: ACTION : U
STATE: (5, 2) :: ACTION : U
STATE: (5, 4) :: ACTION : U
STATE: (5, 8) :: ACTION : U
STATE: (6, 1) :: ACTION : R
STATE: (6, 2) :: ACTION : L
STATE: (6, 4) :: ACTION : R
STATE: (6, 5) :: ACTION : R
STATE: (6, 6) :: ACTION : R
STATE: (6, 7) :: ACTION : U
STATE: (6, 8) :: ACTION : U
STATE: (7, 1) :: ACTION : D
STATE: (7, 2) :: ACTION : D
STATE: (7, 4) :: ACTION : U
STATE: (7, 7) :: ACTION : R
STATE: (7, 8) :: ACTION : L
STATE: (8, 1) :: ACTION : D
STATE: (8, 2) :: ACTION : D
STATE: (8, 4) :: ACTION : R

```
STATE:  (8, 5)   ::   ACTION : R
STATE:  (8, 6)   ::   ACTION : R
STATE:  (8, 7)   ::   ACTION : D
STATE:  (8, 8)   ::   ACTION : D
THE GOAL STATE IS ::  (7, 7)
THE COMPLETE PATH TRAVELLED BY THE AGENT ::  [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (5, 8), (6, 8), (7, 8), (7, 7)]
TOTAL COST OF THE PATH IS ::  12
```

Note: If the number of iterations is more, then the agent makes better decisions as it gets more chance to explore the environment. Here in assignment #, no iteration is taken as 100.