a) Theory:

i. Policy Design:

We use **Q-learning** to train an epsilon-greedy agent to find the shortest path between position (0, 0) to a randomly located goal in a 2D rectangular grid in the 2D GridWorld environment of size (8, 8) (Can be altered). The agent starts at (1,1) always and is restricted to displacing itself up/down/left/right by 1 grid square per action. The agent cannot move through walls (painted in blue). The agent gets teleported to the *power* position when it steps on a *portal.* The goal location can take random location values ranging from (5,5) upto (8,8) excluding forbidden areas like walls, portals and power. The agent exploration parameter epsilon also decays by a multiplicative constant after every training episode from 1 to 0.1. It does not decay further as the agent requires some randomness to figure out the random changes in the goal location. On every episode, the agent restarts from (1,1).

Q-learning prefers the best Q-value from the next state. Our *epsilon-greedy* agent would randomly choose an action from the allowed actions with a probability of epsilon. Otherwise, it looks into its knowledge base Q and chooses the best action (with maximum Q-value) from the state-action pairs. Thus it is not bothered by the actual policy that is being followed (**Off-policy TD control**).

The formula used for computing Q-values is as follows:

$$Q[s,\ a]\ <-\ Q[s,a]\ +\ \alpha\,[\,R[s,a]\ +\ \gamma\ *\ max_{a'}(Q[s,a'])\ -\ Q[s,a]]$$

where Q[s,a] are the Q-values and $\alpha$is the learning parameter, R[s,a] is the corresponding reward value, $\gamma$is the discount factor and a' is the action leading to the next state. In the submitted code, the learning rate α and discount factor $\gamma$are set to 0.99, also the exploration probability ϵstarts from 1 and decays till 0.636 (experimental value).

Following is the code implementing the above logic:

```
   def train(self, memory):
       # ----------------------------
       # Update:
       #
       # Q[s,a] <- Q[s,a] + alpha * (R[s,a] + gamma * max(Q[s,:]) -
Q[s,a])
       #
       #  R[s,a] = reward for taking action a from state s
       #  alpha = learning rate
       #  gamma = discount factor
       # ----------------------------
       (state, action, state_next, reward, _) = memory
```

```
        sa = state + (action,)
        self.Q[sa] += self.alpha * (reward +
self.gamma*np.max(self.Q[state_next]) - self.Q[sa])
```

ii. Reward Function Design:

The reward function is implemented as a table of state-action pairs of shape (rows * cols * 4) i.e. (Number of rows on grid * Number of columns on grid * Number of actions[Up/down/left/right]), each cell holding the reward value. The state-actions leading towards goal (four instances) are assigned to as goal reward i.e. 100. The remaining cells are assigned to as non-goal rewards/penalties i.e. -0.1 (minus to facilitate shortest path search. The reward function helps the agent to learn. There is no optimal policy followed here. The agent greedily hurries towards the goal location. The negative rewards act as penalties and for every step, the reward value of the agent keeps decreasing. The reward table is an instance of the AgentWorld class.

The _build_rewards function creates the rewards table:

```
def _build_rewards(self, goal_loc):
        # Define agent rewards R[s,a]
        r_goal = 100  # reward for arriving at terminal state (bottom-right
corner)
        r_nongoal = -0.1  # penalty for not reaching terminal state
        R = r_nongoal * np.ones(self.state_dim + self.action_dim,
dtype=float)  # R[s,a] Initialization
        gx, gy = goal_loc
        if gy > 0:
            R[gx, gy - 1, self.action_dict["Down"]] = r_goal  # arrive from
above
        if gx > 0:
            R[gx - 1, gy, self.action_dict["Right"]] = r_goal  # arrive
from the right
        if gx < self.Nx - 1:
            R[gx + 1, gy, self.action_dict["Left"]] = r_goal  # arrive from
the left
        if gy < self.Ny - 1:
            R[gx, gy + 1, self.action_dict["Up"]] = r_goal  # arrive from
the below
        return R
```

The R table is a read-only table which is referred to at different points in the code.

iii. Knowledge Base:

The Q table is implemented as a table of state-action pairs of shape (rows * cols * 4) i.e. (Number of rows on grid * Number of columns on grid * Number of actions[Up/down/left/right]). It is initialized to 0. As the training proceeds, the Q-values are updated based on the individual steps. The Q table is an instance of the Agent class. It is called by the agent when:

1. Fetching an action to execute in the get_action function at every iteration of every episode.
2. Updating the current state-action value in the train function at every iteration of every episode.
3. After every episode to display the knowledge base in the mainloop function.
4. At the end of all episodes to print the knowledge base.

1. get_action function:

```
def get_action(self, env):
        # Epsilon-greedy agent policy
        if random.uniform(0, 1) < self.epsilon:
            # explore
            return np.random.choice(env.allowed_actions())
        else:
            # exploit on allowed actions
            state = env.agent
            actions_allowed = env.allowed_actions()
            Q_s = self.Q[state[0], 7 - state[1], actions_allowed]
            actions_greedy = actions_allowed[np.flatnonzero(Q_s ==
np.max(Q_s))]
            return np.random.choice(actions_greedy)
```

2. train function mentioned above.

3.
```
            logging.debug('='*30+f" episode {episode+1}/{N_episodes}
"+'='*30+f'\neps = {agent.epsilon:.3F}\n \
            path followed = {path}\npath_cost = {reward_episode:.1F}\n
optimal_path = {optimal_paths[self.goal_loc]}\n \
                optimal cost = {cost_optimal[self.goal_loc]}\nKnowledge
base: {agent.Q}')
```

4. display_results function:

```
def display_result(self, env):
        # Clear the game scene
```

```
        env.canvas.delete("all")
        # Fetch the preferred directions in the final episode
        greedy_policy = -1 * np.ones((self.state_dim[0],
self.state_dim[1]), dtype=int)
        for x in range(self.state_dim[0]):
            for y in range(self.state_dim[1]):
                if (x, y) not in env.walls:
                    if self.Q[x, y, :].any():
                        Qi = np.flatnonzero(self.Q[x, y, :])
                        temp = np.full_like(self.Q[x, y, :], -np.inf)
                        temp[Qi] = self.Q[x, y, :][Qi]
                        greedy_policy[x, y] = np.argmax(temp)
                        # Visualization of preferred direction in the final
episode
                        x1, y1 = x, y
                        x1 = x1 * row_h
                        y1 = y1 * col_w
                        x2 = x1 + row_h
                        y2 = y1 + col_w
                        env.canvas.create_rectangle(x1, y1, x2, y2,
fill=Green_color, tags='final_op')
                        env.canvas.create_text(x1 + row_h//2, y1 +
col_w//2, text=f'{list(env.action_dict.keys())[greedy_policy[x, y]]}',
tags='final_op')

        print("\nGreedy policy(y, x):")
        print(np.transpose(greedy_policy))
        print()
        while(True):
            if not env.toggle:
                env.window.update()
```

b) Output of first 3 iterations

In all the episodes, the agent has reached the goal, sooner or later. Here for every episode, the exploration parameter eps, **path followed**, **path cost**, **optimal path** for the same goal position as in this episode and its corresponding **optimal cost** and **knowledge base** are printed.

```
============================== episode 1/500
==============================
eps = 0.990
            path followed = [(1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1,
1), (1, 0), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (2, 3), (3, 3), (2,
3), (1, 3), (0, 3), (1, 3), (0, 3), (0, 4), (0, 3), (1, 3), (0, 3), (0,
2), (0, 1), (0, 0), (1, 0), (1, 1), (0, 1), (0, 2), (0, 3), (0, 2), (0,
1), (0, 2), (0, 1), (0, 0), (1, 0), (0, 0), (0, 1), (0, 0), (1, 0), (0,
0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 3), (0, 2), (0,
3), (0, 4), (0, 5), (0, 6), (1, 6), (1, 5), (7, 4), (7, 5), (7, 6), (6,
6), (7, 6), (6, 6), (6, 7), (6, 6), (6, 7), (7, 7), (6, 7), (5, 7), (4,
7), (3, 7), (3, 6), (7, 4), (7, 3), (7, 4), (7, 3), (6, 3), (5, 3), (5,
2), (5, 1), (5, 0), (5, 1), (5, 0), (6, 0), (5, 0), (4, 0), (3, 0), (2,
0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 0), (3, 1), (3, 0), (4, 0), (3,
0), (4, 0), (5, 0), (6, 0), (7, 0), (7, 4), (7, 5), (7, 6), (6, 6), (6,
7), (6, 6), (6, 5)]
path_cost = 89.4
optimal_path = [(1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (1, 0),
(1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (2, 3), (3, 3), (2, 3), (1, 3),
(0, 3), (1, 3), (0, 3), (0, 4), (0, 3), (1, 3), (0, 3), (0, 2), (0, 1),
(0, 0), (1, 0), (1, 1), (0, 1), (0, 2), (0, 3), (0, 2), (0, 1), (0, 2),
(0, 1), (0, 0), (1, 0), (0, 0), (0, 1), (0, 0), (1, 0), (0, 0), (1, 0),
(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 3), (0, 2), (0, 3), (0, 4),
(0, 5), (0, 6), (1, 6), (1, 5), (7, 4), (7, 5), (7, 6), (6, 6), (7, 6),
(6, 6), (6, 7), (6, 6), (6, 7), (7, 7), (6, 7), (5, 7), (4, 7), (3, 7),
(3, 6), (7, 4), (7, 3), (7, 4), (7, 3), (6, 3), (5, 3), (5, 2), (5, 1),
(5, 0), (5, 1), (5, 0), (6, 0), (5, 0), (4, 0), (3, 0), (2, 0), (1, 0),
(2, 0), (3, 0), (3, 1), (3, 0), (3, 1), (3, 0), (4, 0), (3, 0), (4, 0),
(5, 0), (6, 0), (7, 0), (7, 4), (7, 5), (7, 6), (6, 6), (6, 7), (6, 6),
(6, 5)]
                optimal cost = 89.40000000000002
Knowledge base: [[[ 0.         0.         0.         0.       ]
 [-0.099      0.         0.         0.       ]
 [-0.099      0.         0.         0.       ]
 [-0.0999999  0.         0.         0.       ]
 [-0.0999999  0.        -0.09999   -0.0999999]
 [-0.0999999  0.        -0.0999999  0.       ]
 [-0.0999999  0.        -0.0999999 -0.099    ]
 [ 0.         0.        -0.0999999 -0.1      ]]
```

```
[[ 0.          0.          0.          0.        ]
 [ 0.         -0.099       0.          0.        ]
 [ 0.          0.         -0.099       0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.09999     0.         -0.099     ]
 [ 0.          0.          0.          0.        ]
 [-0.09999    -0.099       0.          0.        ]
 [ 0.         -0.1        -0.099      -0.099     ]]

[[ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.         -0.09999   ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.099       0.          0.        ]
 [ 0.         -0.099       0.         -0.099     ]]

[[ 0.          0.          0.         -0.099     ]
 [ 0.          0.         -0.099       0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [-0.099      -0.099       0.          0.        ]
 [-0.099       0.          0.          0.        ]
 [-0.09999    -0.099       0.          0.        ]
 [ 0.         -0.099      -0.09999    -0.09999   ]]

[[ 0.          0.          0.         -0.099     ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.09999     0.         -0.099     ]]

[[ 0.          0.          0.         -0.099     ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
```

```
 [ 0.          0.          0.         -0.099    ]
 [ 0.          0.         -0.099      0.       ]
 [-0.099       0.         -0.099      0.       ]
 [ 0.         -0.099      -0.09999   -0.099    ]]

[[-0.0999999   0.          0.         -0.099    ]
 [ 0.          0.         -0.09999   -0.0999999]
 [ 0.          0.         99.         0.       ]
 [ 0.          0.          0.         0.       ]
 [ 0.          0.          0.        -0.099    ]
 [ 0.          0.          0.         0.       ]
 [ 0.          0.          0.         0.       ]
 [ 0.         -0.09999     0.         0.       ]]

[[ 0.         -0.099       0.         0.       ]
 [-0.09999    -0.099       0.         0.       ]
 [-0.09999     0.          0.         0.       ]
 [-0.099       0.         -0.099     -0.09999  ]
 [ 0.          0.         -0.09999    0.       ]
 [ 0.          0.          0.         0.       ]
 [ 0.          0.          0.         0.       ]
 [ 0.         -0.099       0.         0.       ]]]

2021-05-19 16:35:25,606
============================== episode 2/500
==============================
eps = 0.980
            path followed = [(0, 1), (0, 2), (0, 1), (0, 2), (0, 3), (0,
4), (0, 3), (0, 2), (0, 1), (0, 0), (0, 1), (0, 2), (0, 1), (0, 2), (0,
1), (0, 0), (0, 1), (0, 2), (0, 1), (1, 1), (0, 1), (1, 1), (0, 1), (0,
2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 5), (1, 5), (7, 4), (7, 3), (7,
2), (7, 3), (6, 3), (6, 2), (7, 2), (6, 2), (5, 2), (5, 1), (5, 2), (5,
3), (6, 3), (5, 3), (6, 3), (5, 3), (6, 3), (7, 3), (6, 3), (6, 2), (6,
3), (5, 3), (6, 3), (5, 3), (5, 2), (6, 2), (5, 2), (5, 3), (6, 3), (6,
2), (5, 2), (5, 3), (5, 2), (6, 2), (6, 3), (7, 3), (7, 4), (7, 3), (7,
2), (7, 3), (7, 4), (7, 3), (7, 2), (7, 3), (7, 4), (7, 5), (6, 5), (5,
5), (6, 5), (7, 5), (7, 6)]
path_cost = 92.0
optimal_path = [(0, 1), (0, 2), (0, 1), (0, 2), (0, 3), (0, 4), (0, 3),
(0, 2), (0, 1), (0, 0), (0, 1), (0, 2), (0, 1), (0, 2), (0, 1), (0, 0),
```

```
(0, 1), (0, 2), (0, 1), (1, 1), (0, 1), (1, 1), (0, 1), (0, 2), (0, 3),
(0, 4), (0, 5), (0, 6), (0, 5), (1, 5), (7, 4), (7, 3), (7, 2), (7, 3),
(6, 3), (6, 2), (7, 2), (6, 2), (5, 2), (5, 1), (5, 2), (5, 3), (6, 3),
(5, 3), (6, 3), (5, 3), (6, 3), (7, 3), (6, 3), (6, 2), (6, 3), (5, 3),
(6, 3), (5, 3), (5, 2), (6, 2), (5, 2), (5, 3), (6, 3), (6, 2), (5, 2),
(5, 3), (5, 2), (6, 2), (6, 3), (7, 3), (7, 4), (7, 3), (7, 2), (7, 3),
(7, 4), (7, 3), (7, 2), (7, 3), (7, 4), (7, 5), (6, 5), (5, 5), (6, 5),
(7, 5), (7, 6)]
                optimal cost = 92.00000000000001
Knowledge base: [[[ 0.         0.         0.         0.       ]
 [-0.09999    0.         0.         0.       ]
 [-0.09999    0.        -0.099      0.       ]
 [-0.1        0.         0.         0.       ]
 [-0.1        0.        -0.0999999 -0.0999999]
 [-0.1        0.        -0.1        0.       ]
 [-0.1        0.        -0.1       -0.0999999]
 [ 0.         0.        -0.1       -0.1      ]]

[[ 0.         0.         0.         0.       ]
 [ 0.        -0.099      0.         0.       ]
 [ 0.        -0.099     -0.099      0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.        -0.09999    0.        -0.099    ]
 [ 0.         0.         0.         0.       ]
 [-0.09999   -0.0999999  0.         0.       ]
 [ 0.        -0.1       -0.099     -0.099    ]]

[[ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.        -0.09999  ]
 [ 0.         0.         0.         0.       ]
 [ 0.        -0.099      0.         0.       ]
 [ 0.        -0.099      0.        -0.099    ]]

[[ 0.         0.         0.        -0.099    ]
 [ 0.         0.        -0.099      0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
```

```
 [-0.099     -0.099      0.          0.       ]
 [-0.099      0.         0.          0.       ]
 [-0.09999   -0.099      0.          0.       ]
 [ 0.        -0.099     -0.09999   -0.09999  ]]

[[ 0.         0.         0.        -0.099    ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.        -0.09999    0.        -0.099    ]]

[[ 0.         0.         0.        -0.099    ]
 [ 0.         0.         0.         0.       ]
 [ 0.         0.         0.        -0.099    ]
 [ 0.         0.         0.         0.       ]
 [-0.0999999  0.         0.        -0.1      ]
 [-0.099      0.        -0.0999999 -0.0999999]
 [-0.099      0.        -0.09999    0.       ]
 [ 0.        -0.099     -0.09999   -0.099    ]]

[[-0.0999999  0.         0.        -0.099    ]
 [ 0.         0.        -0.09999   -0.0999999]
 [ 0.        96.9309    99.        96.9309   ]
 [ 0.         0.         0.         0.       ]
 [-0.09999   -0.1        0.        -0.0999999]
 [ 0.        -0.09999   -0.0999999 -0.099    ]
 [ 0.         0.         0.         0.       ]
 [ 0.        -0.09999    0.         0.       ]]

[[ 0.        -0.099      0.         0.       ]
 [98.9990001 -0.099      0.         0.       ]
 [-0.0999999 -0.099      0.         0.       ]
 [-0.1        0.        -0.09999   -0.09999  ]
 [-0.0999999 -0.09999   -0.1        0.       ]
 [ 0.        -0.099     -0.0999999  0.       ]
 [ 0.         0.         0.         0.       ]
 [ 0.        -0.099      0.         0.       ]]]
```

```
2021-05-19 16:35:25,717
============================== episode 3/500
==============================
eps = 0.970
            path followed = [(0, 1), (0, 0), (1, 0), (0, 0), (0, 1), (0,
0), (0, 1), (1, 1), (1, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 0), (1,
0), (2, 0), (3, 0), (2, 0), (2, 1), (3, 1), (3, 2), (3, 3), (3, 2), (3,
1), (2, 1), (2, 0), (1, 0), (2, 0), (1, 0), (1, 1), (0, 1), (0, 2), (0,
3), (0, 4), (1, 4), (0, 4), (0, 5), (1, 5), (7, 4), (7, 5), (7, 4), (7,
3), (6, 3), (5, 3), (6, 3), (7, 3), (7, 4), (7, 5), (7, 6), (6, 6), (7,
6), (7, 7), (6, 7), (6, 6), (7, 6), (7, 7), (6, 7), (7, 7), (6, 7), (6,
6), (6, 5), (7, 5), (6, 5), (7, 5), (7, 4), (7, 3), (7, 4), (7, 5), (7,
6), (7, 7), (6, 7), (6, 6), (6, 7), (5, 7)]
path_cost = 92.7
optimal_path = [(0, 1), (0, 0), (1, 0), (0, 0), (0, 1), (0, 0), (0, 1),
(1, 1), (1, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 0), (1, 0), (2, 0),
(3, 0), (2, 0), (2, 1), (3, 1), (3, 2), (3, 3), (3, 2), (3, 1), (2, 1),
(2, 0), (1, 0), (2, 0), (1, 0), (1, 1), (0, 1), (0, 2), (0, 3), (0, 4),
(1, 4), (0, 4), (0, 5), (1, 5), (7, 4), (7, 5), (7, 4), (7, 3), (6, 3),
(5, 3), (6, 3), (7, 3), (7, 4), (7, 5), (7, 6), (6, 6), (7, 6), (7, 7),
(6, 7), (6, 6), (7, 6), (7, 7), (6, 7), (7, 7), (6, 7), (6, 6), (6, 5),
(7, 5), (6, 5), (7, 5), (7, 4), (7, 3), (7, 4), (7, 5), (7, 6), (7, 7),
(6, 7), (6, 6), (6, 7), (5, 7)]
                optimal cost = 92.70000000000002
Knowledge base: [[[ 0.          0.          0.          0.        ]
 [-0.09999     0.          0.          0.        ]
 [-0.0999999   0.         -0.099       0.        ]
 [-0.1         0.          0.         -0.099     ]
 [-0.1         0.         -0.0999999  -0.0999999 ]
 [-0.1         0.         -0.1         0.        ]
 [-0.1         0.         -0.1        -0.1       ]
 [ 0.          0.         -0.1        -0.1       ]]

[[ 0.          0.          0.          0.        ]
 [ 0.         -0.099       0.          0.        ]
 [ 0.         -0.09999    -0.099       0.        ]
 [ 0.         -0.099       0.          0.        ]
 [ 0.         -0.09999     0.         -0.099     ]
 [ 0.          0.          0.          0.        ]
```

```
 [-0.0999999  -0.1         0.          0.        ]
 [ 0.         -0.1        -0.09999    -0.1       ]]

[[ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.         -0.09999   ]
 [ 0.          0.          0.          0.        ]
 [-0.099      -0.099       0.         -0.099     ]
 [ 0.         -0.1        -0.099      -0.09999   ]]

[[ 0.          0.          0.         -0.099     ]
 [ 0.          0.         -0.099       0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [-0.09999    -0.099       0.          0.        ]
 [-0.09999     0.         -0.099       0.        ]
 [-0.09999    -0.09999    -0.099       0.        ]
 [ 0.         -0.09999    -0.09999    -0.09999   ]]

[[ 0.          0.          0.         -0.099     ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.09999     0.         -0.099     ]]

[[ 0.          0.          0.         98.99901   ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.         -0.099     ]
 [ 0.          0.          0.          0.        ]
 [-0.0999999   0.          0.         -0.1       ]
 [-0.099       0.         -0.0999999  -0.0999999 ]
 [-0.099       0.         -0.09999     0.        ]
 [ 0.         -0.099      -0.09999    -0.099     ]]

[[-0.1         0.          0.         -0.1       ]
```

```
 [ 0.          0.         -0.1        -0.1       ]
 [ 0.         96.9309     97.9209     96.84258309]
 [ 0.          0.          0.          0.        ]
 [-0.09999    -0.1         0.         -0.1       ]
 [ 0.         -0.09999    -0.0999999  -0.099     ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.09999     0.          0.        ]]

[[-0.0999999  -0.09999     0.          0.        ]
 [96.85150289 96.83101693  0.          0.        ]
 [-0.1        -0.0999999   0.          0.        ]
 [-0.1         0.         -0.1        -0.09999   ]
 [-0.0999999  -0.0999999  -0.1         0.        ]
 [ 0.         -0.099      -0.0999999   0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.099       0.          0.        ]]]
```

c) Final output

Here the optimal path shown is the path covered in some previous episode with the goal in the same place as this episode.

```
============================== episode 500/500
==============================
eps = 0.636
            path followed = [(1, 0), (2, 0), (2, 1), (1, 1), (0, 1),
(1, 1), (0, 1), (1, 1), (0, 1), (1, 1), (0, 1), (1, 1), (2, 1), (1,
1), (0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (1, 5), (7, 4), (7, 5),
(7, 6), (6, 6), (6, 5), (7, 5), (7, 6), (7, 5), (7, 6), (7, 7), (7,
6), (7, 7), (7, 6), (7, 5), (7, 4), (7, 3), (6, 3), (6, 2), (5, 2),
(5, 3), (5, 2), (6, 2), (5, 2), (5, 3), (5, 2), (6, 2), (7, 2), (7,
3), (7, 4), (7, 3), (6, 3), (6, 2), (6, 3), (6, 2), (5, 2), (5, 3),
(5, 2), (6, 2), (5, 2), (6, 2), (5, 2), (6, 2), (6, 3), (5, 3), (6,
3), (5, 3), (5, 2), (6, 2), (5, 2), (6, 2), (5, 2), (5, 1), (5, 2),
(6, 2), (5, 2), (5, 3), (6, 3), (6, 2), (5, 2), (6, 2), (6, 3), (5,
3), (6, 3), (7, 3), (7, 2), (7, 3), (6, 3), (6, 2), (5, 2), (5, 3),
(5, 2), (6, 2), (6, 3), (6, 2), (6, 3), (5, 3), (5, 2), (5, 1), (5,
2), (6, 2), (6, 3), (5, 3), (6, 3), (5, 3), (6, 3), (6, 2), (5, 2),
```

(5, 1), (5, 0), (5, 1), (5, 2), (6, 2), (5, 2), (5, 3), (5, 2), (5, 3), (6, 3), (6, 2), (6, 3), (7, 3), (7, 4), (7, 3), (6, 3), (5, 3), (6, 3), (5, 3), (5, 2), (6, 2), (5, 2), (6, 2), (5, 2), (5, 1), (5, 2), (6, 2), (6, 3), (5, 3), (5, 2), (5, 1), (5, 0), (5, 1), (5, 0), (4, 0), (5, 0), (4, 0), (3, 0), (3, 1), (2, 1), (1, 1), (1, 0), (0, 0), (0, 1), (1, 1), (0, 1), (1, 1), (0, 1), (0, 2), (0, 1), (1, 1), (2, 1), (3, 1), (2, 1), (3, 1), (3, 2), (3, 1), (3, 0), (4, 0), (5, 0), (4, 0), (3, 0), (4, 0), (3, 0), (4, 0), (3, 0), (2, 0), (2, 1), (2, 0), (1, 0), (0, 0), (1, 0), (1, 1), (0, 1), (1, 1), (0, 1), (0, 0), (0, 1), (1, 1), (0, 1), (1, 1), (0, 1), (0, 0), (0, 1), (1, 1), (1, 0), (0, 0), (1, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (2, 1), (3, 1), (2, 1), (3, 1), (2, 1), (1, 1), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3), (3, 2), (3, 1), (2, 1), (1, 1), (1, 0), (0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (2, 1), (1, 1), (2, 1), (3, 1), (3, 0), (4, 0), (3, 0), (4, 0), (3, 0), (2, 0), (2, 1), (1, 1), (0, 1), (1, 1), (2, 1), (2, 0), (2, 1), (3, 1), (2, 1), (1, 1), (1, 0), (2, 0), (2, 1), (2, 0), (3, 0), (3, 1), (2, 1), (1, 1), (0, 1), (1, 1), (0, 1), (1, 1), (2, 1), (2, 0), (1, 0), (0, 0), (0, 1), (1, 1), (0, 1), (1, 1), (1, 0), (2, 0), (3, 0), (2, 0), (2, 1), (2, 0), (3, 0), (4, 0), (3, 0), (2, 0), (3, 0), (3, 1), (2, 1), (2, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (1, 1), (0, 1), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2, 1), (2, 0), (2, 1), (1, 1), (2, 1), (1, 1), (0, 1), (1, 1), (0, 1), (1, 1), (0, 1), (0, 2), (0, 1), (0, 0), (1, 0), (1, 1), (2, 1), (2, 0), (3, 0), (4, 0), (3, 0), (4, 0), (3, 0), (4, 0), (3, 0), (3, 1), (2, 1), (2, 0), (1, 0), (1, 1), (0, 1), (0, 2), (0, 1), (0, 0), (1, 0), (2, 0), (3, 0), (2, 0), (3, 0), (2, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (5, 0), (4, 0), (5, 0), (4, 0), (3, 0), (3, 1), (2, 1), (1, 1), (0, 1), (1, 1), (1, 0), (0, 0), (1, 0), (0, 0), (1, 0), (1, 1), (1, 0), (0, 0), (0, 1), (1, 1), (0, 1), (1, 1), (0, 1), (0, 2), (0, 3), (0, 4), (0, 3), (1, 3), (0, 3), (0, 4), (0, 5), (0, 4), (1, 4), (1, 5), (7, 4), (7, 3), (6, 3), (6, 2), (6, 3), (6, 2), (7, 2), (7, 3), (6, 3), (5, 3), (5, 2), (5, 3), (6, 3), (5, 3), (5, 2), (6, 2), (5, 2), (5, 3), (6, 3), (5, 3), (6, 3), (6, 2), (5, 2), (6, 2), (5, 2), (6, 2), (5, 2), (5, 3), (6, 3), (7, 3), (7, 4), (7, 5), (7, 4), (7, 5), (6, 5), (7, 5), (7, 4), (7, 5), (7, 4), (7, 3), (6, 3), (7, 3), (6, 3), (5, 3), (5, 2), (6, 2), (5, 2), (5, 1), (5, 2), (5, 3), (5, 2), (6, 2), (6, 3), (7, 3), (7, 4), (7, 5), (6, 5), (7, 5), (6, 5), (6, 6), (6, 5), (7, 5), (7, 4), (7, 3), (7, 2), (7, 3), (7, 2), (6, 2), (6, 3), (6, 2), (7, 2),

```
(7, 3), (7, 2), (7, 3), (6, 3), (5, 3), (6, 3), (6, 2), (7, 2), (7,
3), (6, 3), (5, 3), (5, 2), (6, 2), (7, 2), (7, 3), (7, 4), (7, 5),
(7, 6), (7, 5), (7, 6), (7, 5), (7, 4), (7, 5), (6, 5), (6, 6), (6,
5), (6, 6), (6, 5), (6, 6), (6, 5), (6, 6), (6, 5), (6, 6), (7, 6),
(7, 7), (7, 6), (7, 7), (7, 6), (7, 7), (7, 6), (6, 6), (6, 7), (6,
6), (6, 5), (6, 6), (6, 7), (6, 6), (6, 7), (5, 7), (6, 7), (7, 7),
(7, 6), (7, 5), (6, 5), (7, 5), (7, 4), (7, 3), (7, 2), (7, 3), (6,
3), (6, 2), (7, 2), (7, 3), (6, 3), (5, 3), (6, 3), (7, 3), (7, 2),
(6, 2), (7, 2), (6, 2), (7, 2), (7, 3), (7, 2), (7, 3), (7, 4), (7,
5), (7, 4), (7, 5), (7, 6), (7, 7), (7, 6), (7, 5), (6, 5), (6, 6),
(6, 7), (7, 7), (7, 6), (7, 7), (7, 6), (6, 6), (6, 5), (6, 6), (6,
7), (5, 7), (4, 7), (3, 7), (3, 6), (7, 4), (7, 5), (7, 6), (7, 5),
(7, 6), (7, 7), (6, 7), (5, 7), (6, 7), (7, 7), (7, 6), (7, 7), (6,
7), (5, 7), (6, 7), (5, 7), (6, 7), (6, 6), (6, 7), (5, 7), (6, 7),
(6, 6), (7, 6), (7, 5), (7, 6), (6, 6), (6, 5), (7, 5), (6, 5), (5,
5)]
path_cost = 42.6
optimal_path = [(0, 1), (0, 0), (1, 0), (1, 1), (1, 0), (2, 0), (3,
0), (4, 0), (5, 0), (4, 0), (5, 0), (4, 0), (5, 0), (6, 0), (7, 0),
(7, 4), (7, 5), (6, 5), (5, 5)]
                optimal cost = 98.2
Knowledge base: [[[-9.90000000e-02  0.00000000e+00  0.00000000e+00
-9.99900000e-02]
 [-1.00000000e-01  0.00000000e+00 -9.99900000e-02 -9.99999990e-02]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]]

[[-9.99900000e-02 -9.90000000e-02  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -9.90000000e-02  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]
```

```
[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]

[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00  0.00000000e+00 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]

[[ 0.00000000e+00  1.82191064e+03  0.00000000e+00  1.82360599e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  2.65398650e+03  0.00000000e+00  2.68136014e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]]

[[ 0.00000000e+00  1.47587079e+03  0.00000000e+00  1.54784171e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  2.33260888e+03  0.00000000e+00  2.45973588e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00  0.00000000e+00 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]

[[ 1.93970945e+03  1.92059838e+03  0.00000000e+00  1.93990639e+03]
 [ 1.27050316e+03  0.00000000e+00  1.27025095e+03  1.27037769e+03]
 [ 0.00000000e+00  1.39009917e+03  1.37609817e+03  1.37609817e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

```
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00  0.00000000e+00]]

[[ 1.41632343e+03  1.43058668e+03  0.00000000e+00  0.00000000e+00]
 [ 1.12298110e+03  1.13431093e+03  1.13419536e+03  0.00000000e+00]
 [ 1.04877098e+03  1.04887791e+03  1.04877098e+03  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00  0.00000000e+00]]]
```

Since my model did not train well i.e. for the second row training is not taking place, the final output above is not the optimal output. So attaching the result of an optimal episode also.

```
============================== episode 227/500
==============================
eps = 0.636
            path followed = [(1, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0,
4), (0, 5), (1, 5), (7, 4), (7, 3), (7, 4), (7, 5), (6, 5)]
path_cost = 98.8
optimal_path = [(1, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5),
(1, 5), (7, 4), (7, 3), (7, 4), (7, 5), (6, 5)]
                optimal cost = 98.8
Knowledge base: [[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00
-9.90000000e-02]
 [-1.00000000e-01  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -9.99999000e-02  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]]

[[-9.90000000e-02 -9.90000000e-02  0.00000000e+00  0.00000000e+00]
```

```
 [ 0.00000000e+00 -9.99999990e-02 -9.90000000e-02  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01 -9.99999990e-02  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]

[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]

[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00  0.00000000e+00 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]

[[ 0.00000000e+00  1.28510510e+03  0.00000000e+00  1.54138348e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.75854015e+03  0.00000000e+00  1.75854017e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]]

[[ 0.00000000e+00  9.18644309e+02  0.00000000e+00  1.15443293e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.28588831e+03  0.00000000e+00  1.29897813e+03]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00  0.00000000e+00 -1.00000000e-01]
```

```
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]]

[[ 1.87756198e+03  1.77198350e+03  0.00000000e+00  1.81558097e+03]
 [ 1.07210581e+03  0.00000000e+00  1.08303617e+03  1.07210688e+03]
 [ 0.00000000e+00  9.07645512e+02  8.98560812e+02  9.97568065e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.00000000e-01 -1.00000000e-01  0.00000000e+00 -1.00000000e-01]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01 -1.00000000e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00  0.00000000e+00]]

[[ 9.10510545e+02  9.10603507e+02  0.00000000e+00  0.00000000e+00]
 [ 8.98380561e+02  8.89279364e+02  7.15457312e+02  0.00000000e+00]
 [ 9.70947209e+02  9.70948200e+02  9.80855767e+02  0.00000000e+00]
 [-1.00000000e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]
 [-1.00000000e-01 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01 -1.00000000e-01  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.00000000e-01  0.00000000e+00  0.00000000e+00]]]
```

References

1. https://github.com/ankonzoid/LearningX/tree/master/classical_RL/gridworld
2. https://github.com/aqeelanwar/Snake-And-Apple
3. https://towardsdatascience.com/making-simple-games-in-python-f35f3ae6f31a
4.