CSBB301: Artificial Intelligence Lab

LAB 10: MDP and Reinforcement Learning

Date Assigned : 26/10/2023 Date Submitted : 1/11/2023

Submitted By: Rohit Kumar

Roll No: 211210053

Branch: CSE

Semester: 5th Sem

Group : 2

Submitted To: Dr. Chandra Prakash

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

PART A : Markov Decision Process (MDP) [10]

10.1 A Markov decision process (MDP) refers to a stochastic decision-making process that uses a mathematical framework to model the decision-making of a dynamic system. It is used in scenarios where the results are either random or controlled by a decision maker, which makes sequential decisions over time. Suppose that we have three states: sunny, cloudy, and rainy. The probability of changing the weather is represented as follows. In case it's sunny right now, there's an 80% probability that it will be sunny tomorrow, 10% chance of cloudiness, and 10% chance of rain. Likewise, if it's cloudy right now, there's a 40% chance it will stay cloudy, a 40% chance it will go sunny, and a 20% chance it will start to rain.

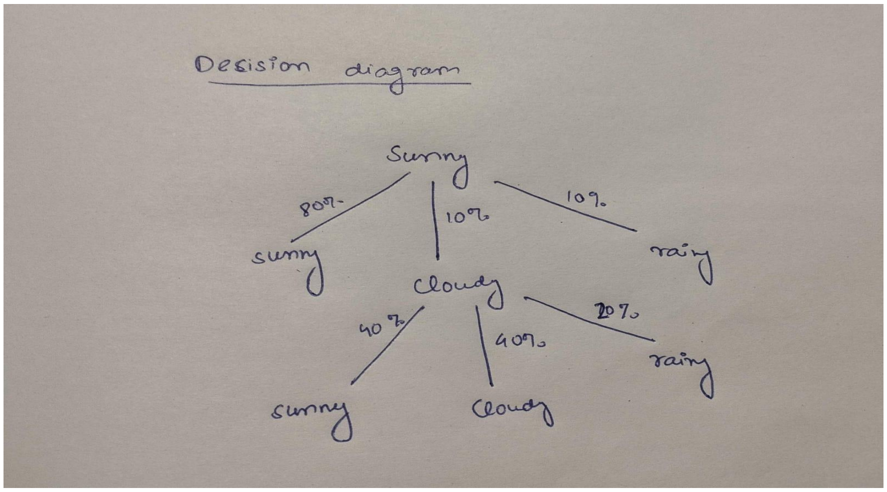
a) Convert this information as a transition matrix and represent it using a decision diagram.

b) Use the random.choices() function to choose the starting state randomly based on the starting probabilities. Write a Program to generate a sequence of 10 states using the transition matrix, and print out the sequence of states as they are generated.

Transition Matrix (P):

	SUNNY	CLOUDY	RAINY
SUNNY	0.8	0.1	0.1
CLOUDY	0.4	0.4	0.2
RAINY	0	0	1

Decision Diagram



CODE:

```python
import random
import numpy as np

# Transition matrix
transition_matrix = np.array([
    [0.8, 0.1, 0.1],  # Sunny -> [Sunny, Cloudy, Rainy]
    [0.4, 0.4, 0.2],  # Cloudy -> [Sunny, Cloudy, Rainy]
    [0, 0, 1]    # Rainy -> [Sunny, Cloudy, Rainy]
])

# Create a dictionary to map state names to row indices
state_to_index = {"Sunny": 0, "Cloudy": 1, "Rainy": 2}

# Starting probabilities
start_probabilities = [0.6, 0.3, 0.1]  # Sunny, Cloudy, Rainy

# Choose the initial state based on starting probabilities
initial_state = random.choices(["Sunny", "Cloudy", "Rainy"], start_probabilities)[0]

# Generate a sequence of 10 states
sequence = [initial_state]
current_state = initial_state
for _ in range(9):
    next_state = random.choices(["Sunny", "Cloudy", "Rainy"], transition_matrix[state_to_index[current_state]])[0]
    sequence.append(next_state)
    current_state = next_state

# Print the sequence of states
print("Generated sequence of states:")
print(sequence)
```

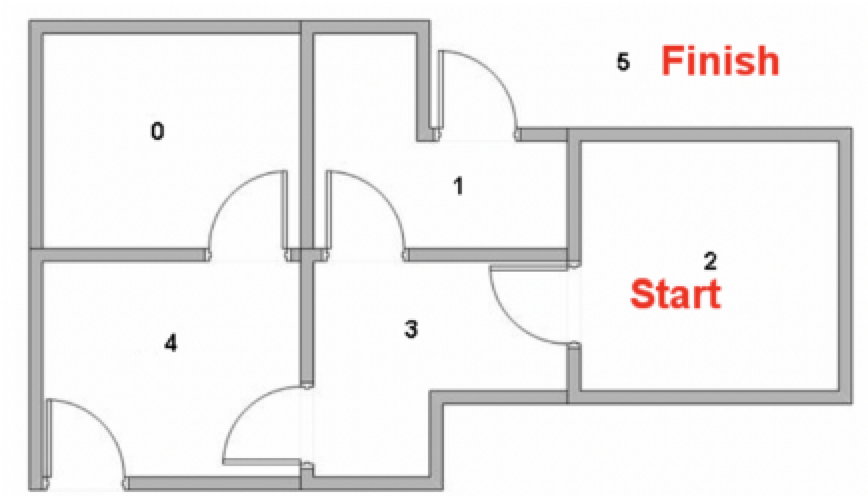
```
Generated sequence of states:
['Rainy', 'Rainy', 'Rainy', 'Rainy', 'Rainy', 'Rainy', 'Rainy', 'Rainy', 'Rainy', 'Rainy']
```

```python
import numpy as np

# Transition probabilities
transition_matrix = np.array([
    [0.8, 0.1, 0.1],  # Sunny -> [Sunny, Cloudy, Rainy]
    [0.4, 0.4, 0.2],  # Cloudy -> [Sunny, Cloudy, Rainy]
    [0, 0, 1]    # Rainy -> [Sunny, Cloudy, Rainy]
])

print(transition_matrix)
```

```
[[0.8 0.1 0.1]
 [0.4 0.4 0.2]
 [0.  0.  1. ]]
```

PART B : Reinforcement Learning : PathFinder Bot

10.2 As discussed in the class suppose we have 5 rooms A to E, in a building connected by certain doors :
We can consider outside of the building as one big room say F to cover the building. There are two
doors lead to the building from F, that is through room B and room E.



Step 1: Modeling the environment-

- Represent the rooms by graph,
- Each room as a vertex (or node) and
- Each door as an edge (or link).
- Goal room is the node F

Goal : Outside the building : Node F Assign Reward Value to each room

State: Each room (including outside building)

Action : Agent's Movement from 1 room to next room

Initial state : C (random)

Reward: Goal Node :highest reward (100) rest – 0;

```python
import numpy as np
```

```python
# R matrix

Rewards = np.matrix([[0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 100.],
        [0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 100.],
        [0., 100., 0., 0., 100., 100.]])

Rewards
```

```
matrix([[  0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0., 100.],
        [  0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0., 100.],
        [  0., 100.,   0.,   0., 100., 100.]])
```

```python
# Q matrix: zero matrix  of size same as R matrix

Q = np.matrix([[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., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.]])
Q
```

```
matrix([[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., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.]])
```

```python
# Gamma (learning parameter).
gamma = 0.8
```

```python
# Initial state. (Usually to be chosen at random)
initial_state = 2

# Write your Code to choose random State
import random

# List of states
states = [0,1,2,3,4,5]
# Choose a random state
random_state = random.choice(states)

# Print the random state
print( random_state)
```

```
3
```

```python
# This function returns all available actions in the state given as an argument
def available_actions(state):
    current_state_row = Rewards[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act
```

```python
# Get available actions in the current state
available_act = available_actions(initial_state)
```

```python
# This function chooses at random which action to be performed within the range
# of all the available actions.
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action
```

```python
# Sample next action to be performed
action = sample_next_action(available_act)
```

```python
# This function updates the Q matrix according to the path selected and the Q
# learning algorithm
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index] # WRITE YOUR CODE HERE

    # Q learning formula
    Q[current_state, action] = Rewards[current_state, action] + gamma * max_value

# Update Q matrix
update(initial_state,action,gamma)
```

```python
#------------------------------------------------------------------------------
# Training

# Train over 10 000 iterations. (Re-iterate the process above).
for i in range(10000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)# WRITE YOUR CODE HERE )
    action = sample_next_action(available_act)# WRITE YOUR CODE HERE )
    score= update(current_state,action,gamma)

    # The "trained" Q matrix
print("The Trained Q matrix:")
print(Q)


# Normalize the "trained" Q matrix
print("Trained Normalized Q matrix:")
Q_nor=(Q - np.min(Q)) / (np.max(Q) - np.min(Q))# WRITE YOUR CODE HERE
print(Q_nor), i
```

```
The Trained Q matrix:
[[320. 400. 320. 320. 400. 400.]
 [320. 400. 320. 320. 400. 500.]
 [320. 400. 320. 320. 400. 400.]
 [320. 400. 320. 320. 400. 400.]
 [320. 400. 320. 320. 400. 500.]
 [320. 500. 320. 320. 500. 500.]]
Trained Normalized Q matrix:
[[0.         0.44444444 0.         0.         0.44444444 0.44444444]
 [0.         0.44444444 0.         0.         0.44444444 1.        ]
 [0.         0.44444444 0.         0.         0.44444444 0.44444444]
 [0.         0.44444444 0.         0.         0.44444444 0.44444444]
 [0.         0.44444444 0.         0.         0.44444444 1.        ]
 [0.         1.         0.         0.         1.         1.        ]]
(None, 9999)
```

```
#----------------------------------------------------------------------------------
# Testing

#STATES = [A,B,C,D,E,F]
#nO_State=[0,1,2,3,4,5]

# Goal state = 5
# Best sequence path starting from 2 -> 2, 3, 1, 5

current_state = 2
steps = [current_state]

while current_state != 5:

    next_step_index = np.where(Q[current_state,] == np.max(Q[current_state,]))[1]

    if next_step_index.shape[0] > 1:
        next_step_index = int(np.random.choice(next_step_index, size = 1))
    else:
        next_step_index = int(next_step_index)

    steps.append(next_step_index)
    current_state = next_step_index
```

```
# Print selected sequence of steps
print("Selected path:")
print(steps)# WRITE YOUR CODE HERE)
```

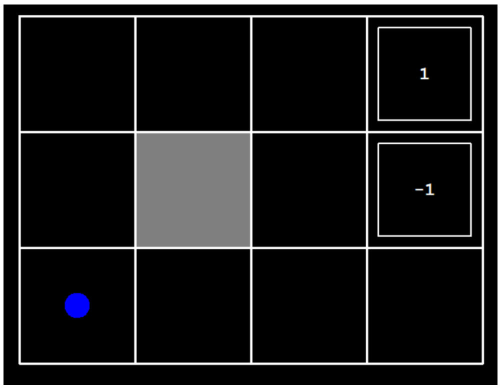
```
Selected path:
[2, 5]
```

PART C : Reinforcement Learning in Pacman [45 Marks]

MDPs

- **python gridworld.py -m** // This runs the Gridworld environment to visualize and experiment with Markov Decision Processes (MDPs).

```
E:\lab10>python gridworld.py -m
## Disabling Agents in Manual Mode (-m) ##

RUNNING 1 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: north
Ended in state: (0, 0)
```



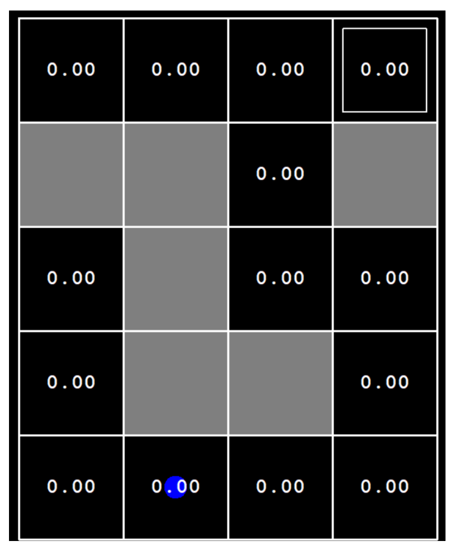

```
Started in state: (3, 2)
Took action: exit
Ended in state: TERMINAL_STATE
Got reward: 1

EPISODE 1 COMPLETE: RETURN WAS 0.43046721000000016


AVERAGE RETURNS FROM START STATE: 0.43046721000000016
```

- **python gridworld.py -h** // This help showing information for the Gridworld environment.

```
E:\lab10>python gridworld.py -h
Usage: gridworld.py [options]

Options:
  -h, --help            show this help message and exit
  -d DISCOUNT, --discount=DISCOUNT
                        Discount on future (default 0.9)
  -r R, --livingReward=R
                        Reward for living for a time step (default 0.0)
  -n P, --noise=P       How often action results in unintended direction
                        (default 0.2)
  -e E, --epsilon=E     Chance of taking a random action in q-learning
                        (default 0.3)
  -l P, --learningRate=P
                        TD learning rate (default 0.5)
  -i K, --iterations=K  Number of rounds of value iteration (default 10)
  -k K, --episodes=K    Number of epsiodes of the MDP to run (default 1)
  -g G, --grid=G        Grid to use (case sensitive; options are BookGrid,
                        BridgeGrid, CliffGrid, MazeGrid, default BookGrid)
  -w X, --windowSize=X  Request a window width of X pixels *per grid cell*
                        (default 150)
  -a A, --agent=A       Agent type (options are 'random', 'value' and 'q',
                        default random)
  -t, --text            Use text-only ASCII display
  -p, --pause           Pause GUI after each time step when running the MDP
  -q, --quiet           Skip display of any learning episodes
  -s S, --speed=S       Speed of animation, S > 1.0 is faster, 0.0 < S < 1.0
                        is slower (default 1.0)
  -m, --manual          Manually control agent
  -v, --valueSteps      Display each step of value iteration
```

- **python gridworld.py -g MazeGrid** // Run Gridworld with the MazeGrid layout.

```
E:\lab10>python gridworld.py -g MazeGrid

RUNNING 1 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: west
Ended in state: (0, 0)
Got reward: 0.0

Started in state: (0, 0)
Took action: east
Ended in state: (1, 0)
Got reward: 0.0
```



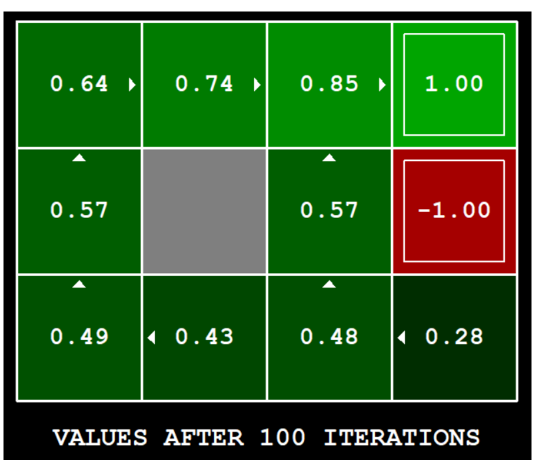
Question 10.3.1 (10 points): Value Iteration

- **python autograder.py -q q1** //Run autograder for the question about Value Iteration.

```
E:\lab10>python autograder.py -q q1
E:\lab10\autograder.py:17: DeprecationWarning:
  import imp
Starting on 10-28 at 17:35:48

Question q1
===========

*** PASS: test_cases\q1\1-tinygrid.test
*** PASS: test_cases\q1\2-tinygrid-noisy.test
*** PASS: test_cases\q1\3-bridge.test
*** PASS: test_cases\q1\4-discountgrid.test

### Question q1: 4/4 ###


Finished at 17:35:48

Provisional grades
==================
Question q1: 4/4
------------------
Total: 4/4
```

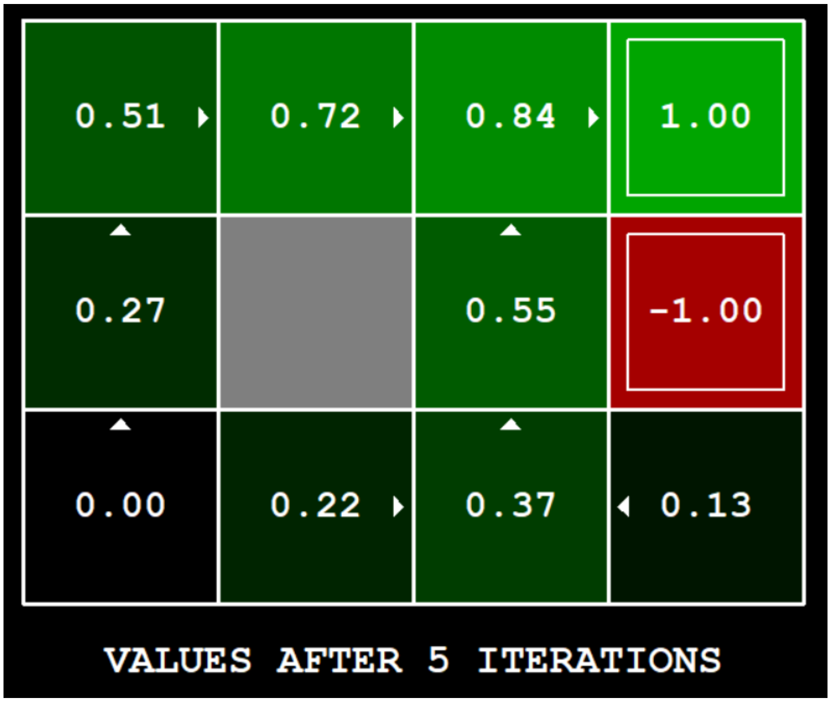
- **python gridworld.py -a value -i 100 -k 10** //Run the Gridworld environment with value iteration for 100 iterations and a convergence limit of 10.

```
E:\lab10>python gridworld.py -a value -i 100 -k 10

RUNNING 10 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: north
Ended in state: (0, 0)
Got reward: 0.0

Started in state: (0, 0)
Took action: north
```



- **python gridworld.py -a value -i 5** // Run the Gridworld environment with 5 iterations

```
E:\lab10>python gridworld.py -a value -i 5

RUNNING 1 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: north
Ended in state: (0, 1)
Got reward: 0.0

Started in state: (0, 1)
Took action: north
Ended in state: (0, 2)
Got reward: 0.0

Started in state: (0, 2)
Took action: east
Ended in state: (1, 2)
Got reward: 0.0

Started in state: (1, 2)
Took action: east
Ended in state: (2, 2)
Got reward: 0.0

Started in state: (2, 2)
Took action: east
Ended in state: (3, 2)
Got reward: 0.0

Started in state: (3, 2)
Took action: exit
Ended in state: TERMINAL_STATE
Got reward: 1

EPISODE 1 COMPLETE: RETURN WAS 0.5904900000000002


AVERAGE RETURNS FROM START STATE: 0.5904900000000002
```



Question 10.3.2 (5 point): Bridge Crossing Analysis

- **python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.2** // Running Gridworld with the BridgeGrid layout using value iteration, with specific discount and noise settings.

```
E:\lab10>python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.2

RUNNING 1 EPISODES

BEGINNING EPISODE: 1

Started in state: (1, 1)
Took action: west
Ended in state: (1, 0)
Got reward: 0.0

Started in state: (1, 0)
Took action: exit
Ended in state: TERMINAL_STATE
Got reward: -100

EPISODE 1 COMPLETE: RETURN WAS -90.0


AVERAGE RETURNS FROM START STATE: -90.0
```

- **python autograder.py -q q2** //Running the autograder for the Bridge Crossing Analysis question.

```
E:\lab10>python autograder.py -q q2
E:\lab10\autograder.py:17: DeprecationWarnin
  import imp
Starting on 10-28 at 17:41:00

Question q2
===========

*** PASS: test_cases\q2\1-bridge-grid.test

### Question q2: 1/1 ###


Finished at 17:41:00

Provisional grades
==================
Question q2: 1/1
------------------
Total: 1/1
```

Question 10.3.3 (10 points): Policies

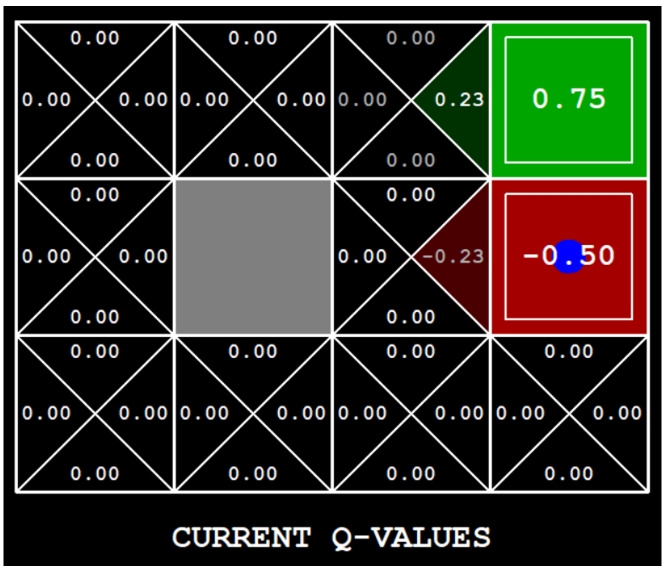
- **python autograder.py -q q3 //**Running the autograder for the question related to policies.

```
E:\lab10>python autograder.py -q q3
E:\lab10\autograder.py:17: DeprecationWarning:
  import imp
Starting on 10-28 at 17:41:36

Question q3
===========

*** PASS: test_cases\q3\1-question-3.1.test
*** PASS: test_cases\q3\2-question-3.2.test
*** PASS: test_cases\q3\3-question-3.3.test
*** PASS: test_cases\q3\4-question-3.4.test
*** PASS: test_cases\q3\5-question-3.5.test

### Question q3: 5/5 ###


Finished at 17:41:37

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5
```

Question 10.3.4 (10 points): Q-Learning

- **python gridworld.py -a q -k 5 -m** //Running Gridworld with Q-learning for 5 episodes.

```
E:\lab10>python gridworld.py -a q -k 5 -m

RUNNING 5 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: east
Ended in state: (1, 0)
Got reward: 0.0

Started in state: (1, 0)
Took action: east
Ended in state: (1, 0)
```



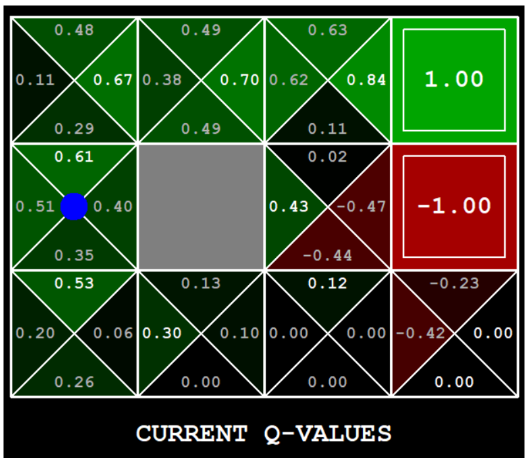
- **python autograder.py -q q4** //Running the autograder for the Q-Learning question.

```
E:\lab10>python autograder.py -q q4
E:\lab10\autograder.py:17: DeprecationWarning:
  import imp
Starting on 10-28 at 17:45:18

Question q4
===========

*** PASS: test_cases\q4\1-tinygrid.test
*** PASS: test_cases\q4\2-tinygrid-noisy.test
*** PASS: test_cases\q4\3-bridge.test
*** PASS: test_cases\q4\4-discountgrid.test

### Question q4: 1/1 ###


Finished at 17:45:18

Provisional grades
==================
Question q4: 1/1
------------------
Total: 1/1
```

Question 10.3.5(5 points): Epsilon Greedy

- **python gridworld.py -a q -k 100** //Running Gridworld with Q-learning for 100 episodes.

```
E:\lab10>python gridworld.py -a q -k 100

RUNNING 100 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: west
Ended in state: (0, 0)
Got reward: 0.0

Started in state: (0, 0)
```



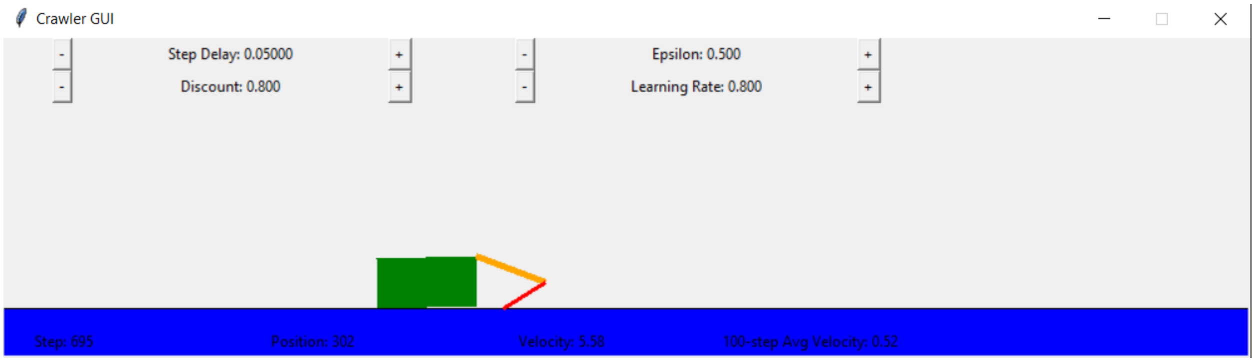
- **python autograder.py -q q5** //Running the autograder for the Epsilon Greedy question.

```
E:\lab10>python autograder.py -q q5
E:\lab10\autograder.py:17: DeprecationWarning:
  import imp
Starting on 10-28 at 17:48:57

Question q5
===========

*** PASS: test_cases\q5\1-tinygrid.test
*** PASS: test_cases\q5\2-tinygrid-noisy.test
*** PASS: test_cases\q5\3-bridge.test
*** PASS: test_cases\q5\4-discountgrid.test

### Question q5: 3/3 ###


Finished at 17:48:58

Provisional grades
==================
Question q5: 3/3
------------------
Total: 3/3
```

 # Q-learning crawler robot:

- **python crawler.py** //Running a Q-learning crawler robot, which is a separate project or simulation involving a crawling robot trained using Q-learning.

```
E:\lab10>python crawler.py
```

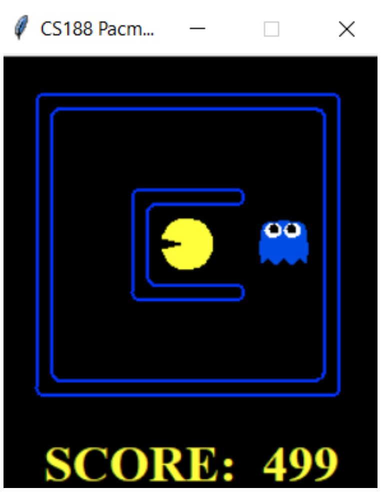


Question 10.3.6 (5 point): Q-Learning and Pacman

- **python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid** //Running Pacman with a Q-learning agent for a certain number of games on the smallGrid layout.

```
E:\lab10>python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
Beginning 2000 episodes of Training
Reinforcement Learning Status:
        Completed 100 out of 2000 training episodes
        Average Rewards over all training: -500.11
        Average Rewards for last 100 episodes: -500.11
        Episode took 0.97 seconds
Reinforcement Learning Status:
        Completed 200 out of 2000 training episodes
```

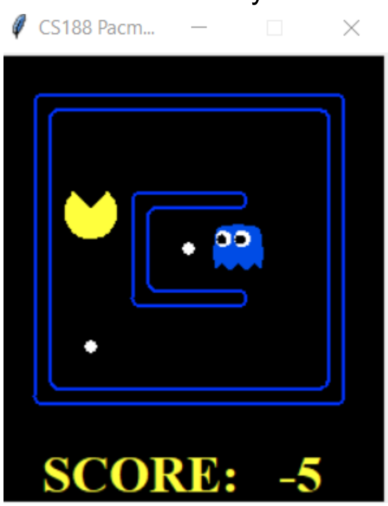
```
        Episode took 2.50 seconds
Training Done (turning off epsilon and alpha)
---------------------------------------------
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Average Score: 497.8
Scores:        495.0, 499.0, 499.0, 503.0, 495.0, 499.0, 499.0, 499.0, 495.0, 495.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```



- **python autograder.py -q q6** //Running the autograder for the Q-Learning and Pacman question..

```
E:\lab10>python autograder.py -q q6
E:\lab10\autograder.py:17: DeprecationWarr
  import imp
Starting on 10-28 at 17:53:23

Question q6
===========

*** PASS: test_cases\q6\grade-agent.test

### Question q6: 1/1 ###


Finished at 17:53:23

Provisional grades
==================
Question q6: 1/1
------------------
Total: 1/1
```

- **python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10**
  //Running Pacman with the PacmanQAgent for evaluation with 10 training games on the smallGrid layout.




```
E:\lab10>python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
Beginning 10 episodes of Training
Pacman died! Score: -508
Pacman died! Score: -528
Pacman died! Score: -505
Pacman died! Score: -507
Pacman died! Score: -517
Pacman died! Score: -506
Pacman died! Score: -512
Pacman died! Score: -511
Pacman died! Score: -520
Pacman died! Score: -509
Training Done (turning off epsilon and alpha)
---------------------------------------------
Average Score: -512.3
Scores:        -508.0, -528.0, -505.0, -507.0, -517.0, -506.0, -512.0, -511.0, -520.0, -509.0
Win Rate:      0/10 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
```

Submission and Evaluation:

- **python autograder.py** //Running the autograder for the entire project to assess your code.

```
Provisional grades
==================
Question q1: 4/4
Question q2: 1/1
Question q3: 5/5
Question q4: 1/1
Question q5: 3/3
Question q6: 1/1
Question q7: 1/1
Question q8: 3/3
------------------
Total: 19/19
```