# Task 7.1D: Function approximation implementation

**GitHub Link:**

**Objective:** To implement Task 1.1P with the following methods:

- Semi-Gradient Sarsa(0) (From Slide 14)
- Semi-Gradient TD(λ) (From Slide 9)

In [1]:
```
#Loading all of our Libraries...
import numpy as np
import matplotlib.pyplot as plt
# import gym
import sys
```

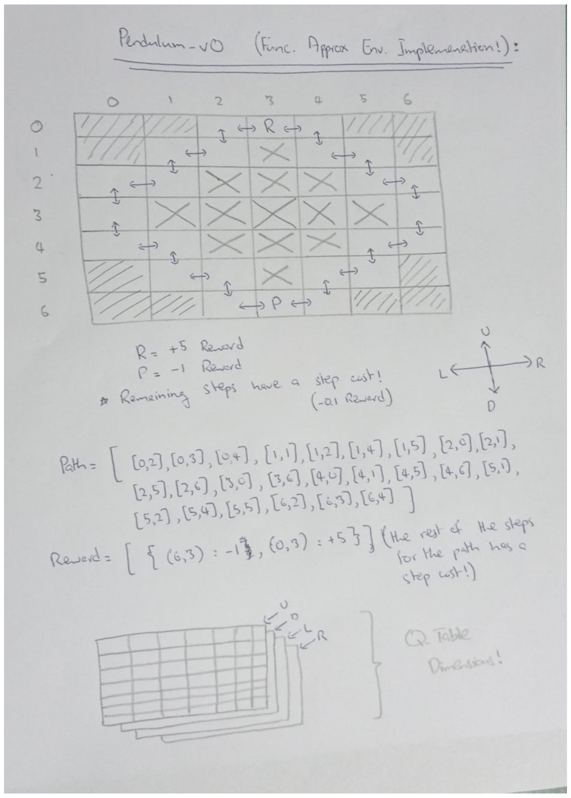
In [2]:
```
#Connecting our Google Drive...
from google.colab import drive

drive.mount('/content/drive')
sys.path.insert(0,'/content/drive/MyDrive/Colab Notebooks/')

#Importing our GridWorld Module after connection...
from GW import Grid, print_values, print_policy
```

```
Mounted at /content/drive
```

## Creating our Environment

In [3]:
```
#All the Constants...
ALL_POSSIBLE_ACTIONS = ('U', 'D', 'L', 'R')
num_episodes = 100
GAMMA = 0.9
ALPHA = 0.1
eps = 0.1
```

This is environment we have specified to the current model. This time we discretized the model based on the GridWorld. We were able to create our own Custom GridWorld based on the circular path of the Pendulum. Here's how we have developed our Environment on Sketch:


Pendulum-v0 (Func. Approx Env. Implementation!):
0 1 2 3 4 5 6
0 1 2 3 4 5 6
R
P
R = +5 Reward
P = -1 Reward
* Remaining steps have a step cost! (-0.1 Reward)
U
D
L
R
Path = [ [0,2], [0,3], [0,4], [1,1], [1,2], [1,4], [1,5], [2,0], [2,1], [2,5], [2,6], [3,0], [3,6], [4,0], [4,1], [4,5], [4,6], [5,1], [5,2], [5,4], [5,5], [6,2], [6,3], [6,4] ]
Reward = [ { (6,3) : -1, (0,3) : +5 } ] (the rest of the steps for the path has a step cost!)
U
D
L
R
Q Table Dimensions!

In [4]:
```
#Creating the Pendulum Environment...

pendulum = Grid(7, 7, (6, 3))
step_cost = -0.1

#Dictionary of the rewards assigned at every step of the path...
rewards = {
    (0, 3): 5,
    (0, 2): step_cost,
    (0, 4): step_cost,
    (1, 1): step_cost,
    (1, 2): step_cost,
    (1, 4): step_cost,
    (1, 5): step_cost,
    (2, 0): step_cost,
    (2, 1): step_cost,
    (2, 5): step_cost,
    (2, 6): step_cost,
    (3, 0): step_cost,
    (3, 6): step_cost,
    (4, 0): step_cost,
    (4, 1): step_cost,
    (4, 5): step_cost,
    (4, 6): step_cost,
    (5, 1): step_cost,
    (5, 2): step_cost,
    (5, 4): step_cost,
    (5, 5): step_cost,
    (6, 2): step_cost,
    (6, 4): step_cost,
    (6, 3): -1
    }

#Dictionary of the actions assigned at every step of the path...
actions = {
    (0, 2): ('D', 'R'),
    (0, 3): ('L', 'R'),
    (0, 4): ('L', 'D'),
    (1, 1): ('R', 'D'),
    (1, 2): ('L', 'U'),
    (1, 4): ('R', 'U'),
    (1, 5): ('L', 'D'),
    (2, 0): ('D', 'R'),
    (2, 1): ('L', 'U'),
    (2, 5): ('R', 'U'),
    (2, 6): ('L', 'D'),
    (3, 0): ('U', 'D'),
    (3, 6): ('U', 'D'),
    (4, 0): ('U', 'R'),
    (4, 1): ('L', 'D'),
    (4, 5): ('R', 'D'),
    (4, 6): ('L', 'U'),
    (5, 1): ('U', 'R'),
    (5, 2): ('L', 'D'),
    (5, 4): ('D', 'R'),
    (5, 5): ('L', 'U'),
    (6, 2): ('U', 'R'),
    (6, 3): ('L', 'R'),
    (6, 4): ('L', 'U'),
  }

#Setting our Grid with the rewards and actions...
pendulum.set(rewards, pendulum)
```

In [5]:
```
print("------------------------Rewards per state in the Environment------------------------\n")
print_values(rewards, pendulum)
```

```
------------------------Rewards per state in the Environment------------------------

---------------------------
 0.00 |
 0.00 |
-0.10 |
 5.00 |
-0.10 |
 0.00 |
 0.00 |

---------------------------
 0.00 |
-0.10 |
-0.10 |
 0.00 |
-0.10 |
-0.10 |
 0.00 |

---------------------------
-0.10 |
-0.10 |
 0.00 |
 0.00 |
 0.00 |
-0.10 |
-0.10 |

---------------------------
-0.10 |
 0.00 |
 0.00 |
 0.00 |
 0.00 |
 0.00 |
-0.10 |

---------------------------
-0.10 |
-0.10 |
 0.00 |
 0.00 |
 0.00 |
-0.10 |
-0.10 |

---------------------------
 0.00 |
-0.10 |
-0.10 |
 0.00 |
-0.10 |
-0.10 |
 0.00 |

---------------------------
 0.00 |
 0.00 |
-0.10 |
-1.00 |
-0.10 |
 0.00 |
 0.00 |
```

In [6]:
```python
#Enlisting all the possible actions per state...

state = list(actions.keys())
possible_actions = list(actions.values())

print("------------------------Possible Actions at every step in the Environment------------------------\n")
for i in range(len(state)):
  print("{} | {}".format(state[i], possible_actions[i]))
```

```
------------------------Possible Actions at every step in the Environment------------------------

(0, 2) | ('D', 'R')
(0, 3) | ('L', 'R')
(0, 4) | ('L', 'D')
(1, 1) | ('R', 'D')
(1, 2) | ('L', 'U')
(1, 4) | ('R', 'U')
(1, 5) | ('L', 'D')
(2, 0) | ('D', 'R')
(2, 1) | ('L', 'U')
(2, 5) | ('R', 'U')
(2, 6) | ('L', 'D')
(3, 0) | ('U', 'D')
(3, 6) | ('U', 'D')
(4, 0) | ('U', 'R')
(4, 1) | ('L', 'D')
(4, 5) | ('R', 'D')
(4, 6) | ('L', 'U')
(5, 1) | ('U', 'R')
(5, 2) | ('L', 'D')
(5, 4) | ('D', 'R')
(5, 5) | ('L', 'U')
(6, 2) | ('U', 'R')
(6, 3) | ('L', 'R')
(6, 4) | ('L', 'U')
```

## Semi-Gradient Sarsa(0)

We will create some basic functionalities for our algorithms while the agent learns in the environment.

In [24]:
```python
# #Function for getting the actions of our Optimal Policy...

# def max_dict(d):
#   # returns the argmax (key) and max (value) from a dictionary
#   max_key = None
#   max_val = float('-inf')
#   for k, v in d.items():
#     if v > max_val:
#       max_val = v
#       max_key = k
#   return max_key, max_val
```

In [25]:
```python
#Greedy/Exploration Function...

def epsilon_greedy_action(state, epsilon):
    #Exploration
    if np.random.uniform() < epsilon:
        return np.random.choice(ALL_POSSIBLE_ACTIONS)  #Returns letter, not the index...
    #Expoitation
    else:
        _ , maxarg_a = actions[(state[0], state[1])]
        return maxarg_a                                  #Returns letter, not the index...
```

In [26]:
```python
#Function for mapping actions into index...
def action_map(a):
  if a == 'U':
      i = 0
  elif a == 'D':
      i = 1
  elif a == 'R':
      i = 2
  elif a == 'L':
      i = 3
  return i
```

In [27]:
```python
#Function to return reward with from a certain state with Grid Properties...

def step_function(s,a):
  # r = pendulum.move(a)
  i,j = s

  if s in actions.keys():
    if a == 'U':
      i -= 1
    elif a == 'D':
      i += 1
    elif a == 'R':
      j -= 1
    elif a == 'L':
      j += 1

    next_s = (i,j)
    if next_s in actions.keys():
      r = rewards.get(next_s, 0)
      #print(next_s,"-----",r)
      return next_s, r

    else:
      r = 0
      #print(next_s,"-----",r)

      #Undo the move...
      if a == 'U':
        i -= 1
      elif a == 'D':
        i += 1
      elif a == 'R':
        j -= 1
      elif a == 'L':
        j += 1
      #print("\nOut of bounds. Move Undone...")
      next_s = s
      return next_s, r

  else:
    r = 0
    #print("\nOut of bounds. Move Undone...")
    return s, r

#Running a small test for the function...
next_S, r = step_function((5,3),'D')
print(next_S, r)
```

```
(5, 3) 0
```

In [28]:
```python
#Function for computing the gradient of the model...
def grad(Q, state, action):
    ci, cj = state
    gradient = np.zeros_like(Q)
    gradient = 1
    return gradient
```

For our Semi-Gradient SARSA(0), this is the psuedocode for us to implement:

```
Input:
        A differentiable state-action value function $\hat{q}: \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$
        A policy $\pi$ if predicting or $q_\pi$ if estimating (e.g. using $\varepsilon - greedy$)
Algorithm Parameter:
        Step size $\alpha \in (0,1]$
Initialise:
        $\mathbf{w} \in \mathbb{R}^d$ arbitrarily e.g. $\mathbf{w} = 0$
Loop forever (for each episode):
        $S, A \leftarrow$ Initial state and action of episode (e.g. using $\varepsilon - greedy$)
        Loop for each step of the episode until $S \in S^{(\mathrm{Terminal})}$ :
                Take action $A$, observe $R, S'$
                If $S' \in S^{terminal}$ then:
                        $\mathbf{w} = \mathbf{w} + \alpha[R + \gamma\hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$, special case for terminal state can't include future state
                else:
                        Choose $A'$ as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g. using $\varepsilon - greedy$)
                        $\mathbf{w} = \mathbf{w} + \alpha[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$
                        $S \leftarrow S'$
                        $A \leftarrow A'$
```

In [29]:
```python
def semi_gradient_sarsa(num_episodes, alpha, gamma, epsilon):
    total_reward_per_episode = []
    average_reward_per_episode = []

    for i in range(num_episodes):

        #pendulum.set_state(s) / state = env.reset()
        state = (6,3)    #Starting point of the Agent in the Environment...
        num_actions = len(ALL_POSSIBLE_ACTIONS)     # Number of Actions
        Q = np.zeros((7, 7, num_actions))           # Q-table Initialized
        Total_Q = np.zeros((7, 7, num_actions))
        total_reward = 0

        action = epsilon_greedy_action(state, epsilon)
        for t in range(200):

            # Getting next state and reward...
            next_state, reward = step_function(state, action)

            #Getting dimensions of the current and next state in order to update the Q-Table...
            ci, cj = state
            ni, nj = next_state

            #Get the next action...
            next_action = epsilon_greedy_action(next_state, epsilon)

            #Mapping the current and next action...
            a = action_map(action)
            next_a = action_map(next_action)

            #Update the Q-Table...
            td_err = reward + gamma * Q[ni][nj][next_a] - Q[ci][cj][a]
            Q[ci][cj][a] += alpha * td_err * grad(Q[ci][cj][a], state, a)

            #Creating the total sum of the reward...
            total_reward += reward

            #Assign the new state and action and repeat...
            state = next_state
            action = next_action

        #Assigning the final state values in the final step...
        if i + 1 == 100:
            Total_Q += Q

        average_reward_per_episode.append(total_reward/200)
        total_reward_per_episode.append(total_reward)
        print("Episode --- [{}/100]  | Reward ---- {}".format(i + 1, total_reward))

    return total_reward_per_episode, average_reward_per_episode, Total_Q
```

## Semi-Gradient TD($\lambda$)

In [7]:
```python
weights = np.zeros((7, 7, 4))     # Q-table/weights Initialized
weights[3,2,:]
```

Out[7]:
```
array([0., 0., 0., 0.])
```

"Again, we are going to implement some of the basic functionalities for this algorithm as well: "

In [8]:
```python
# Function for predicting the weights...
def predict(state, weights):
  i,j = state
  return np.dot(weights[i,j,:], np.ones(4))
```

In [9]:
```python
#Function to return reward with from a certain state with Grid Properties...

def step_function(s,a):
  # r = pendulum.move(a)
  i,j = s

  if s in actions.keys():
    if a == 0:    #'U'
      i -= 1
    elif a == 1:  #'D'
      i += 1
    elif a == 2:  #'L'
      j -= 1
    elif a == 3:  #'R'
      j += 1

    next_s = (i,j)
    if next_s in actions.keys():
      r = rewards.get(next_s, 0)
      # print(next_s,"-----",r)
      return next_s, r

    else:
      r = 0
      #print(next_s,"-----",r)

      #Undo the move...
      if a == 0:    #'U'
        i += 1
      elif a == 1:  #'D'
        i -= 1
      elif a == 2:  #'L'
        j += 1
      elif a == 3:  #'R'
        j -= 1
      #print("\nOut of bounds. Move Undone...")
      next_s = s
      return next_s, r

  else:
    r = 0
    #print("\nOut of bounds. Move Undone...")
    return s, r
```

In [10]:
```python
#Running a small test for the function...
next_S, r = step_function((0,2),3)
print(next_S, r)
```

```
(0, 3) 5
```

Likewise, for our second algorithm, this is how we will implement from the below pseudocode:

```
Input:
        The policy π to be evaluate
        A differentiable function $\hat{v}: \mathcal{S} \times \mathbb{R}^d \rightarrow \mathbb{R}$
Algorithm Parameter:
        Step size $\alpha \in (0,1]$
        Trace decay rate $\lambda \in [0,1]$
Initialise:
        $\mathbf{w} \in \mathbb{R}^d$ arbitrarily e.g. $\mathbf{w} = 0$
Loop forever (for each episode):
        Initialize $S$
        Reset $\mathbf{z} = \mathbf{0}$
        Loop for each step of the episode until $S \in S^{(\mathrm{Terminal})}$ :
                Choose $A \sim \pi(\cdot \mid S)$
                Take action $A$, observe $R, S'$
                $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S, \mathbf{w})$
                $\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
                $\mathbf{w} = \mathbf{w} + \alpha\delta\mathbf{z}$
                $S \leftarrow S'$
```

In [19]:
```python
def semi_gradient_td_lambda(num_episodes, alpha, gamma, epsilon, lambda_):
        total_reward_per_episode = []
        average_reward_per_episode = []

        for i in range(num_episodes):
            #pendulum.set_state(s) / state = env.reset()
            state = (6,3)    #Starting point of the Agent in the Environment...

            num_actions = len(ALL_POSSIBLE_ACTIONS)     # Number of Actions
            weights = np.zeros((7, 7, num_actions))     # Q-table/weights Initialized
            Total_Q = np.zeros((7, 7, num_actions))
            eligibility_trace = np.zeros((7,7,4))       # Resetting Eligibility Trace

            total_reward = 0
            # a = np.argmax(predict(state, weights))

            for t in range(200):
```

```python
            #Taking a Random Action...
            a = np.random.choice([0,1,2,3])                 # To kick-start the algorithm...

            # Getting next state and reward...
            next_state, reward = step_function(state, a)       #env.step(action)

            #Getting dimensions of the current and next state in order to update the weights...
            ci, cj = state
            # ni, nj = next_state

            #Get the next action...
            next_a = predict(next_state, weights)

            #Computing TD Error (delta)...
            delta = reward + gamma * predict(next_state, weights) - predict(state, weights)

            #Updating the Q-Table/Weights...
            eligibility_trace *= lambda_
            eligibility_trace[ci,cj,int(a)] += 1
            weights += alpha * delta * eligibility_trace

            #Creating the total sum of the reward...
            total_reward += reward

            #Assign the new state and action and repeat...
            state = next_state
            a = next_a

        #Assigning the final state values in the final step...
        if i + 1 == 100:
            Total_Q += weights

        average_reward_per_episode.append(total_reward/200)
        total_reward_per_episode.append(total_reward)
        print("Episode --- [{}/100]  | Reward ---- {}".format(i + 1, total_reward))

    return total_reward_per_episode, average_reward_per_episode, Total_Q
```

## Comparison of Results

### SARSA(0) Results

In [30]:
```python
trpe, sarsa_arpe, Total_Q = semi_gradient_sarsa(num_episodes, ALPHA, GAMMA, eps)
print("Average Reward after 100 Episodes: ",np.mean(trpe))
```

```
Episode --- [1/100]  | Reward ---- -5.3
Episode --- [2/100]  | Reward ---- -4.4
Episode --- [3/100]  | Reward ---- -9.699999999999998
Episode --- [4/100]  | Reward ---- -5.1000000000000005
Episode --- [5/100]  | Reward ---- -8.999999999999998
Episode --- [6/100]  | Reward ---- -7.099999999999999
Episode --- [7/100]  | Reward ---- -9.699999999999996
Episode --- [8/100]  | Reward ---- -11.899999999999983
Episode --- [9/100]  | Reward ---- -16.89999999999997
Episode --- [10/100]  | Reward ---- -6.9999999999999964
Episode --- [11/100]  | Reward ---- -3.500000000000001
Episode --- [12/100]  | Reward ---- -6.199999999999999
Episode --- [13/100]  | Reward ---- -10.399999999999979
Episode --- [14/100]  | Reward ---- -4.3999999999999995
Episode --- [15/100]  | Reward ---- -13.599999999999968
Episode --- [16/100]  | Reward ---- -4.4
Episode --- [17/100]  | Reward ---- -7.699999999999998
Episode --- [18/100]  | Reward ---- -7.1
Episode --- [19/100]  | Reward ---- -5.699999999999999
Episode --- [20/100]  | Reward ---- -12.199999999999973
Episode --- [21/100]  | Reward ---- -6.3999999999999755
Episode --- [22/100]  | Reward ---- -4.299999999999999
Episode --- [23/100]  | Reward ---- -3.3000000000000007
Episode --- [24/100]  | Reward ---- -11.699999999999998
Episode --- [25/100]  | Reward ---- -5.3
Episode --- [26/100]  | Reward ---- -6.299999999999998
Episode --- [27/100]  | Reward ---- -4.8999999999999995
Episode --- [28/100]  | Reward ---- -3.100000000000001
Episode --- [29/100]  | Reward ---- -8.499999999999991
Episode --- [30/100]  | Reward ---- -4.799999999999998
Episode --- [31/100]  | Reward ---- -9.099999999999998
Episode --- [32/100]  | Reward ---- -6.099999999999997
Episode --- [33/100]  | Reward ---- -8.6
Episode --- [34/100]  | Reward ---- -11.299999999999997
Episode --- [35/100]  | Reward ---- -16.69999999999997
Episode --- [36/100]  | Reward ---- -4.2
Episode --- [37/100]  | Reward ---- -11.499999999999975
Episode --- [38/100]  | Reward ---- -4.7
Episode --- [39/100]  | Reward ---- -2.4
Episode --- [40/100]  | Reward ---- -6.799999999999998
Episode --- [41/100]  | Reward ---- -11.399999999999999
Episode --- [42/100]  | Reward ---- -0.8999999999999999
Episode --- [43/100]  | Reward ---- -8.799999999999995
Episode --- [44/100]  | Reward ---- -3.1
Episode --- [45/100]  | Reward ---- -8.499999999999995
Episode --- [46/100]  | Reward ---- -7.6999999999999975
Episode --- [47/100]  | Reward ---- -4.7
Episode --- [48/100]  | Reward ---- -6.799999999999997
Episode --- [49/100]  | Reward ---- -3.200000000000002
Episode --- [50/100]  | Reward ---- -4.3999999999999995
Episode --- [51/100]  | Reward ---- -9.299999999999997
Episode --- [52/100]  | Reward ---- -3.300000000000001
Episode --- [53/100]  | Reward ---- -4.700000000000001
Episode --- [54/100]  | Reward ---- -8.799999999999997
Episode --- [55/100]  | Reward ---- -3.100000000000001
Episode --- [56/100]  | Reward ---- -7.199999999999996
Episode --- [57/100]  | Reward ---- -6.3999999999999995
Episode --- [58/100]  | Reward ---- -4.599999999999999
Episode --- [59/100]  | Reward ---- -8.399999999999997
Episode --- [60/100]  | Reward ---- -4.2
Episode --- [61/100]  | Reward ---- -6.2
Episode --- [62/100]  | Reward ---- -8.099999999999996
Episode --- [63/100]  | Reward ---- -6.1
Episode --- [64/100]  | Reward ---- -2.900000000000001
Episode --- [65/100]  | Reward ---- -6.799999999999998
Episode --- [66/100]  | Reward ---- -7.199999999999998
Episode --- [67/100]  | Reward ---- -6.199999999999999
Episode --- [68/100]  | Reward ---- -7.899999999999998
Episode --- [69/100]  | Reward ---- -2.900000000000001
Episode --- [70/100]  | Reward ---- -4.2
Episode --- [71/100]  | Reward ---- -7.099999999999998
Episode --- [72/100]  | Reward ---- -5.6999999999999975
Episode --- [73/100]  | Reward ---- -10.299999999999995
Episode --- [74/100]  | Reward ---- -11.899999999999983
Episode --- [75/100]  | Reward ---- -4.000000000000001
Episode --- [76/100]  | Reward ---- -8.499999999999995
Episode --- [77/100]  | Reward ---- -7.599999999999998
Episode --- [78/100]  | Reward ---- -6.299999999999997
Episode --- [79/100]  | Reward ---- -4.7
Episode --- [80/100]  | Reward ---- -8.2
Episode --- [81/100]  | Reward ---- -7.499999999999999
Episode --- [82/100]  | Reward ---- -10.299999999999995
Episode --- [83/100]  | Reward ---- -6.199999999999995
Episode --- [84/100]  | Reward ---- -7.499999999999998
Episode --- [85/100]  | Reward ---- -8.099999999999987
Episode --- [86/100]  | Reward ---- -4.000000000000001
Episode --- [87/100]  | Reward ---- -4.6000000000000005
Episode --- [88/100]  | Reward ---- -3.5000000000000004
Episode --- [89/100]  | Reward ---- -4.4
Episode --- [90/100]  | Reward ---- -4.599999999999999
Episode --- [91/100]  | Reward ---- -2.0
Episode --- [92/100]  | Reward ---- -5.499999999999999
Episode --- [93/100]  | Reward ---- -7.499999999999998
Episode --- [94/100]  | Reward ---- -13.999999999999968
Episode --- [95/100]  | Reward ---- -7.499999999999999
Episode --- [96/100]  | Reward ---- -3.1
Episode --- [97/100]  | Reward ---- -12.99999999999997
Episode --- [98/100]  | Reward ---- -8.099999999999998
Episode --- [99/100]  | Reward ---- -8.999999999999996
Episode --- [100/100]  | Reward ---- -5.0
Average Reward after 100 Episodes:  -6.889999999999997
```

In [31]:
```python
print("Average Reward per Episode: \n", sarsa_arpe)
```

```
Average Reward per Episode: 
 [-0.0265, -0.022000000000000002, -0.04849999999999999, -0.025500000000000002, -0.04499999999999999, -0.0355, -0.04849999999999998, -0.05949999999999914, -0.08449999999999985, -0.03499999999999998, -0.017500000000000005, -0.030999999999999996, -0.05199999999999894, -0.022, -0.06799999999999984, -0.022000000000000002, -0.038
49999999999999, -0.0355, -0.02849999999999998, -0.06099999999999986, -0.03199999999999876, -0.02149999999999995, -0.016500000000000004, -0.05849999999999999, -0.0265, -0.03149999999999999, -0.02449999999999997, -0.015500000000000005, -0.04249999999999954, -0.02399999999999999, -0.04549999999999999, -0.03049999999999985,
-0.043, -0.05649999999999999, -0.08349999999999985, -0.021, -0.05749999999999988, -0.0235, -0.012, -0.03399999999999999, -0.05699999999999995, -0.0045, -0.04399999999999998, -0.0155, -0.04249999999999975, -0.03849999999999986, -0.0235, -0.03399999999999999, -0.016000000000000001, -0.022, -0.04649999999999986, -0.0165000000
00000004, -0.023500000000000007, -0.04399999999999984, -0.015500000000000005, -0.03599999999999976, -0.032, -0.02299999999999993, -0.04199999999999998, -0.021, -0.031, -0.04049999999999998, -0.0305, -0.014500000000000004, -0.03399999999999999, -0.03599999999999999, -0.030999999999999996, -0.03949999999999987, -0.014500000
000000004, -0.021, -0.03549999999999999, -0.02849999999999987, -0.05149999999999976, -0.05949999999999914, -0.020000000000000004, -0.04249999999999975, -0.03799999999999999, -0.03149999999999986, -0.0235, -0.04099999999999995, -0.0375, -0.05149999999999976, -0.03099999999999975, -0.03749999999999999, -0.04049999999999999
994, -0.020000000000000004, -0.023000000000000003, -0.0175, -0.022000000000000002, -0.02299999999999993, -0.01, -0.02749999999999997, -0.03749999999999999, -0.06999999999999984, -0.0375, -0.0155, -0.06499999999999985, -0.04049999999999999, -0.04499999999999984, -0.025]
```

In [32]:
```python
print("Q-Table in the 100th Episode: \n", Total_Q)
```

```
Q-Table in the 100th Episode:
 [[[ 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.        ]
  [ 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.        ]
  [ 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.        ]
  [ 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.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.         -0.0199181   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.        ]
  [-0.02999181 -0.00616232 -0.04406767 -0.35249578]
  [ 0.          0.         -0.04935613  0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]]]
```

## TD(Lambda) Results

In [21]:
```python
ALPHA = 0.1  # step size parameter
GAMMA = 1.0  # discount factor
eps = 0.1  # exploration rate
lambda_ = 0.5  # lambda parameter for eligibility trace

trpe_, td_arpe, Total_Q_ = semi_gradient_td_lambda(num_episodes, ALPHA, GAMMA, eps, lambda_)
print("Average Reward after 100 Episodes: ",np.mean(trpe_))
```

```
Episode --- [1/100]  | Reward ---- -5.99999999999997
Episode --- [2/100]  | Reward ---- 12.800000000000024
Episode --- [3/100]  | Reward ---- 5.600000000000028
Episode --- [4/100]  | Reward ---- -15.999999999999986
Episode --- [5/100]  | Reward ---- -4.499999999999984
Episode --- [6/100]  | Reward ---- -13.599999999999978
Episode --- [7/100]  | Reward ---- 7.500000000000025
Episode --- [8/100]  | Reward ---- 1.599999999999996
Episode --- [9/100]  | Reward ---- 23.50000000000001
Episode --- [10/100]  | Reward ---- 4.500000000000018
Episode --- [11/100]  | Reward ---- 3.3000000000000154
Episode --- [12/100]  | Reward ---- -13.699999999999974
Episode --- [13/100]  | Reward ---- 46.499999999999986
Episode --- [14/100]  | Reward ---- -30.600000000000055
Episode --- [15/100]  | Reward ---- -14.49999999999999
Episode --- [16/100]  | Reward ---- -19.20000000000001
Episode --- [17/100]  | Reward ---- 1.1000000000000192
Episode --- [18/100]  | Reward ---- -23.00000000000004
Episode --- [19/100]  | Reward ---- -12.99999999999998
Episode --- [20/100]  | Reward ---- 2.300000000000025
Episode --- [21/100]  | Reward ---- 1.4000000000000017
Episode --- [22/100]  | Reward ---- 10.900000000000013
Episode --- [23/100]  | Reward ---- -15.59999999999997
Episode --- [24/100]  | Reward ---- -11.99999999999998
Episode --- [25/100]  | Reward ---- -17.90000000000001
Episode --- [26/100]  | Reward ---- -29.000000000000032
Episode --- [27/100]  | Reward ---- 39.4999999999999
Episode --- [28/100]  | Reward ---- 33.999999999999986
Episode --- [29/100]  | Reward ---- -15.099999999999982
Episode --- [30/100]  | Reward ---- -21.40000000000003
Episode --- [31/100]  | Reward ---- 0.20000000000001814
Episode --- [32/100]  | Reward ---- 16.500000000000007
Episode --- [33/100]  | Reward ---- -13.899999999999975
Episode --- [34/100]  | Reward ---- -15.899999999999972
Episode --- [35/100]  | Reward ---- -22.40000000000002
Episode --- [36/100]  | Reward ---- 29.99999999999997
Episode --- [37/100]  | Reward ---- 51.49999999999994
Episode --- [38/100]  | Reward ---- -21.00000000000004
Episode --- [39/100]  | Reward ---- -12.19999999999998
Episode --- [40/100]  | Reward ---- 28.700000000000003
Episode --- [41/100]  | Reward ---- 7.700000000000015
Episode --- [42/100]  | Reward ---- -17.90000000000001
Episode --- [43/100]  | Reward ---- -10.399999999999979
Episode --- [44/100]  | Reward ---- -10.899999999999977
Episode --- [45/100]  | Reward ---- -12.299999999999976
Episode --- [46/100]  | Reward ---- -15.399999999999979
Episode --- [47/100]  | Reward ---- -27.50000000000004
Episode --- [48/100]  | Reward ---- -20.8
Episode --- [49/100]  | Reward ---- -12.899999999999972
Episode --- [50/100]  | Reward ---- -12.99999999999997
Episode --- [51/100]  | Reward ---- -12.89999999999998
Episode --- [52/100]  | Reward ---- 18.30000000000002
Episode --- [53/100]  | Reward ---- -12.099999999999977
Episode --- [54/100]  | Reward ---- -10.200000000000031
Episode --- [55/100]  | Reward ---- -17.8
Episode --- [56/100]  | Reward ---- -15.89999999999998
Episode --- [57/100]  | Reward ---- -15.800000000000042
Episode --- [58/100]  | Reward ---- 15.300000000000004
Episode --- [59/100]  | Reward ---- -19.200000000000014
Episode --- [60/100]  | Reward ---- -10.899999999999977
Episode --- [61/100]  | Reward ---- -15.999999999999979
Episode --- [62/100]  | Reward ---- -6.3999999999999755
Episode --- [63/100]  | Reward ---- -20.899999999999984
Episode --- [64/100]  | Reward ---- -14.599999999999975
Episode --- [65/100]  | Reward ---- -17.09999999999999
Episode --- [66/100]  | Reward ---- -4.799999999999976
Episode --- [67/100]  | Reward ---- 10.300000000000027
Episode --- [68/100]  | Reward ---- 2.5000000000000187
Episode --- [69/100]  | Reward ---- -10.899999999999977
Episode --- [70/100]  | Reward ---- -19.800000000000004
Episode --- [71/100]  | Reward ---- -22.10000000000001
Episode --- [72/100]  | Reward ---- 8.000000000000023
Episode --- [73/100]  | Reward ---- 8.200000000000028
Episode --- [74/100]  | Reward ---- -19.30000000000001
Episode --- [75/100]  | Reward ---- -21.5
Episode --- [76/100]  | Reward ---- -16.299999999999976
Episode --- [77/100]  | Reward ---- -18.500000000000004
Episode --- [78/100]  | Reward ---- 9.400000000000022
Episode --- [79/100]  | Reward ---- -18.999999999999996
Episode --- [80/100]  | Reward ---- -13.099999999999973
Episode --- [81/100]  | Reward ---- -25.10000000000003
Episode --- [82/100]  | Reward ---- -11.699999999999978
Episode --- [83/100]  | Reward ---- -18.40000000000001
Episode --- [84/100]  | Reward ---- -27.30000000000005
Episode --- [85/100]  | Reward ---- -17.599999999999998
Episode --- [86/100]  | Reward ---- -14.299999999999969
Episode --- [87/100]  | Reward ---- -22.300000000000047
Episode --- [88/100]  | Reward ---- -21.000000000000036
Episode --- [89/100]  | Reward ---- -30.700000000000056
Episode --- [90/100]  | Reward ---- -12.69999999999997
Episode --- [91/100]  | Reward ---- -7.8999999999999755
Episode --- [92/100]  | Reward ---- -14.099999999999966
Episode --- [93/100]  | Reward ---- -12.399999999999983
Episode --- [94/100]  | Reward ---- -11.799999999999983
Episode --- [95/100]  | Reward ---- 23.399999999999963
Episode --- [96/100]  | Reward ---- 2.100000000000016
Episode --- [97/100]  | Reward ---- -11.80000000000005
Episode --- [98/100]  | Reward ---- 50.79999999999995
Episode --- [99/100]  | Reward ---- -18.0
Episode --- [100/100]  | Reward ---- -22.900000000000023
Average Reward after 100 Episodes:  -6.692999999999997
```

In [22]:
```python
print("Average Reward per Episode: \n", td_arpe)
```

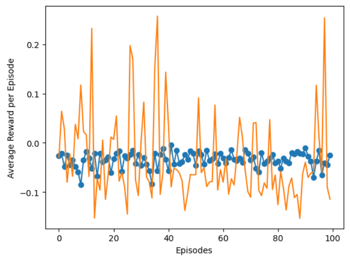
```
Average Reward per Episode: 
 [-0.029999999999999985, 0.06400000000000011, 0.028000000000000014, -0.07999999999999993, -0.022499999999999992, -0.0679999999999999, 0.037500000000000124, 0.007999999999999981, 0.11750000000000005, 0.02250000000000009, 0.016500000000000077, -0.06849999999999987, 0.23249999999999993, -0.15300000000000027, -0.07249999999999995, -
0.09600000000000004, 0.005500000000000096, -0.1150000000000002, -0.06499999999999999, 0.011500000000000126, 0.00700000000000009, 0.05450000000000006, -0.07799999999999985, -0.0599999999999999, -0.08950000000000005, -0.14500000000000016, 0.19749999999999995, 0.16999999999999993, -0.07549999999999991, -0.10700000000000015, 0.001
0000000000000907, 0.08250000000000003, -0.06949999999999988, -0.07949999999999986, -0.1120000000000001, 0.14999999999999986, 0.25749999999999973, -0.10500000000000019, -0.0609999999999999, 0.14350000000000002, 0.038500000000000076, -0.08950000000000005, -0.05199999999999894, -0.05449999999999989, -0.06149999999999988, -0.076
9999999999999, -0.13750000000000002, -0.10400000000000001, -0.06449999999999986, -0.06499999999999985, -0.0644999999999999, 0.0915000000000001, -0.06049999999999988, -0.05100000000000156, -0.0890000000000001, -0.0794999999999999, -0.07900000000000021, 0.07650000000000003, -0.09600000000000007, -0.05449999999999989, -0.079999
99999999989, -0.031999999999999876, -0.10449999999999993, -0.07299999999999987, -0.08549999999999995, -0.023999999999999988, 0.051500000000000136, 0.012500000000000093, -0.05449999999999989, -0.09900000000000002, -0.11050000000000004, 0.04000000000000012, 0.04100000000000014, -0.09650000000000006, -0.1075, -0.0814999999999998
8, -0.09250000000000001, 0.04700000000000011, -0.0949999999999999, -0.06549999999999986, -0.12550000000000014, -0.05849999999999989, -0.09200000000000004, -0.13650000000000026, -0.088, -0.07149999999999984, -0.11150000000000024, -0.10500000000000018, -0.15350000000000028, -0.06349999999999985, -0.03949999999999875, -0.07049
999999999983, -0.06199999999999916, -0.05899999999999914, 0.11699999999999981, 0.0105000000000008, -0.05900000000000254, 0.2539999999999997, -0.09, -0.11450000000000012]
```

In [23]:
```python
print("Q-Table in the 100th Episode: \n", Total_Q_)
```

```
Q-Table in the 100th Episode:
 [[[ 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.        ]
  [ 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.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]]

 [[ 0.         -0.07648783  0.         -0.01323737]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]]

 [[-0.04232098  0.         -0.03281617 -0.10406997]
  [-0.04631232 -0.15434664 -0.07426049 -0.04351224]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]]

 [[ 0.          0.          0.          0.        ]
  [-0.02259149 -0.03279831 -0.01135903 -0.27378823]
  [-0.09638018 -0.60425686  0.00279311 -0.04032602]
  [ 0.          0.          0.          0.        ]
  [-0.03387165 -0.31171903 -0.08025988 -0.01443011]
  [ 0.         -0.00886022 -0.01772045  0.        ]
  [ 0.          0.          0.          0.        ]]

 [[ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.00978951 -0.15654161 -0.12990104 -0.83839656]
  [-0.06156678 -0.08406764 -0.41077049 -0.11807476]
  [-0.08995889 -0.0097468  -0.56850924 -0.09019683]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.          0.        ]]]
```

## Semi-Gradient SARSA(0) vs Semi-Gradient TD(Lambda)

In [33]:
```python
#Plotting the average rewards per episode...
x = [x for x in range(100)]
y1 = sarsa_arpe
y2 = td_arpe

plt.gca()  # shorthand for "get current axis"
plt.plot(x, y1, marker = 'o')
plt.plot(x, y2)
plt.xlabel("Episodes")
plt.ylabel("Average Reward per Episode")
plt.show()
```



## References

- https://www.gymlibrary.dev/environments/classic_control/pendulum/
- https://numpy.org/doc/stable/reference/
- https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
- Semi-Gradient SARSA: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf (p. 152-154)
- Class Slides (Week 7, Week07_01_Function_Approximation1, Slides 9,14)
- https://therenegadecoder.com/code/how-to-plot-a-line-using-matplotlib-in-python/#:~:text=Perhaps%20the%20easiest%20way%20to%20generate%20a%20line,%5B2%2C%204%2C%206%2C%208%2C%2010%5D%20plt.plot%28x%2C%20y%29%20plt.show%28%29
- https://www.w3schools.com/python/matplotlib_markers.asp