CS6700_Assignment3_taxi_part1run

April 22, 2023

0.1 Installation And Imports

[ ]:
```
# installation of older gym for render()
!pip install gym==0.23.1

# some imports and code to ignore deprecation warning
import numpy as np
import random
import gym
import glob
import io
import matplotlib.pyplot as plt
from IPython.display import HTML, display, clear_output
from time import sleep
from tqdm import tqdm
import warnings
import numpy.ma as ma
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting gym==0.23.1
  Downloading gym-0.23.1.tar.gz (626 kB)
                             626.2/626.2 kB
10.3 MB/s eta 0:00:00
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
Requirement already satisfied: gym-notices>=0.0.4 in
/usr/local/lib/python3.9/dist-packages (from gym==0.23.1) (0.0.8)
Requirement already satisfied: cloudpickle>=1.2.0 in
/usr/local/lib/python3.9/dist-packages (from gym==0.23.1) (2.2.1)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.9/dist-
packages (from gym==0.23.1) (1.22.4)
Requirement already satisfied: importlib-metadata>=4.10.0 in
/usr/local/lib/python3.9/dist-packages (from gym==0.23.1) (6.4.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-
packages (from importlib-metadata>=4.10.0->gym==0.23.1) (3.15.0)
```

```
Building wheels for collected packages: gym
  Building wheel for gym (pyproject.toml) … done
  Created wheel for gym: filename=gym-0.23.1-py3-none-any.whl size=701376
sha256=27dce83f56b8acee3076d7794508dc3029304276840f8e7e6deac5cabe265ae7
  Stored in directory: /root/.cache/pip/wheels/4e/be/7e/92a54668db96883e38ce60a9
249dc55de7cd6eee49e7311940
Successfully built gym
Installing collected packages: gym
  Attempting uninstall: gym
    Found existing installation: gym 0.25.2
    Uninstalling gym-0.25.2:
      Successfully uninstalled gym-0.25.2
Successfully installed gym-0.23.1
```

[ ]:
```
# mount drive
from google.colab import drive
drive.mount('/content/drive')
```

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

0.2 Exploring the environment and rendering a manual episode

[ ]:
```
# explore the taxi-v3 environment to know the observation space(states),␣
 ↪actions, rewards etc
# make env and seed it for reproducibility
env = gym.make('Taxi-v3')
env.seed(42)

# number of states
NUM_STATES = env.observation_space.n
print(f'State count - {NUM_STATES}')

# number of actions
NUM_ACTIONS = env.action_space.n
print(f'Action count - {NUM_ACTIONS}')

ACTION_NAMES = ["South", "North", "East", "West", "Pickup", "Dropoff"]
COLOR_NUM_MAPPING = {0 : "Red", 1 : "Green", 2 : "Yellow", 3 : "Blue"}

state = env.reset()
# state decodes as -> (car row, car col, pass loc, dest); pass loc is a color /␣
 ↪4 : "in car"
print(f'Initial state - {list(env.decode(state))}')
env.render()

# STARTING PASSENGER LOCATION -> BLUE COLOR; DEST -> RED COLOR; Taxi -> GREEN␣
 ↪SQUARE
```

```
# Solving the environment manually to understand it better;
for _ in range(3):
    new_state, reward, done, _ = env.step(1)
    env.render()
    print(f'performed {ACTION_NAMES[1]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
new_state, reward, done, _ = env.step(4)
env.render()
print(f'performed {ACTION_NAMES[4]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
for _ in range(2):
    new_state, reward, done, _ = env.step(0)
    env.render()
    print(f'performed {ACTION_NAMES[0]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
for _ in range(4):
    new_state, reward, done, _ = env.step(3)
    env.render()
    print(f'performed {ACTION_NAMES[3]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
for _ in range(2):
    new_state, reward, done, _ = env.step(0)
    env.render()
    print(f'performed {ACTION_NAMES[0]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
new_state, reward, done, _ = env.step(5)
env.render()
print(f'performed {ACTION_NAMES[5]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
```

```
State count - 500
Action count - 6
Initial state - [3, 4, 1, 2]
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | :|
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : :|
| | : | : |
```

```
|Y| : |B: |
+---------+
  (North)
performed North; Current state - [2, 4, 1, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : :_|
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
performed North; Current state - [1, 4, 1, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
performed North; Current state - [0, 4, 1, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
performed Pickup; Current state - [0, 4, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : :_|
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
performed South; Current state - [1, 4, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : :_|
| | : | : |
|Y| : |B: |
+---------+
  (South)
```

```
performed South; Current state - [2, 4, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : :_: |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 3, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : :_: : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 2, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| :_: : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 1, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 0, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (South)
performed South; Current state - [3, 0, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
```

```
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
performed South; Current state - [4, 0, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
performed Dropoff; Current state - [4, 0, 2, 2]; reward : 20; done : True
```

0.3 Plotting and Helper Functions

[ ]:
```
class bcolors:
    RED= '\u001b[31m'
    GREEN= '\u001b[32m'
    RESET= '\u001b[0m'

def see_trained_agent(qtable, stime=1.5):
    episodes_to_preview = 3
    env = gym.make('Taxi-v3')
    env.seed(10)
    opt_func_map = {6: Drive_to_R, 7 : Drive_to_G, 8 : Drive_to_Y, 9 :␣
 ↪Drive_to_B}
    for episode in range(episodes_to_preview):

        # Reset the environment
        state = env.reset()
        clear_output(wait=True); print(f"TRAINED AGENT\n+++++EPISODE␣
 ↪{episode+1}++++++"); env.render()
        eps_reward = 0
        if eps_reward < 0:
            print(f"Score: {bcolors.RED}{eps_reward}{bcolors.RESET}")
        else:
            print(f"Score: {bcolors.GREEN}{eps_reward}{bcolors.RESET}")
        sleep(stime)
        done = False

        while not done:
            # Exploit
            action = np.argmax(qtable[state])
```

```
            # Take an action and observe the reward
            if action < 6:
                next_state, reward, done, info = env.step(action)
                eps_reward += reward
                state = next_state
                clear_output(wait=True); print(f"TRAINED AGENT\n+++++EPISODE␣
 ↪{episode+1}++++++")
                env.render()
                if eps_reward < 0:
                    print(f"Score: {bcolors.RED}{eps_reward}{bcolors.RESET}")
                else:
                    print(f"Score: {bcolors.GREEN}{eps_reward}{bcolors.RESET}")
                sleep(stime)
            else:
                optdone = False
                opt_func = opt_func_map[action]
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    eps_reward += reward
                    _, optdone = opt_func(env, next_state)
                    state = next_state
                    clear_output(wait=True); print(f"TRAINED␣
 ↪AGENT\n+++++EPISODE {episode+1}++++++")
                    env.render()
                    if eps_reward < 0:
                        print(f"Score: {bcolors.RED}{eps_reward}{bcolors.
 ↪RESET}")
                    else:
                        print(f"Score: {bcolors.GREEN}{eps_reward}{bcolors.
 ↪RESET}")
                    sleep(stime)

# function to test an agent after training
def test_agent(q_values, test_episodes):
    env = gym.make('Taxi-v3')
    env.seed(0)
    test_rewards = []
    opt_func_map = {6: Drive_to_R, 7 : Drive_to_G, 8 : Drive_to_Y, 9 :␣
 ↪Drive_to_B} # opt id to option function map
    for _ in tqdm(range(test_episodes)):
        # Reset the environment
        state = env.reset()
        eps_reward = 0
        done = False
        while not done:
```

```
            # Exploit
            action = np.argmax(q_values[state])

            # Take an action and observe the reward
            if action < 6:
                next_state, reward, done, info = env.step(action)
                eps_reward += reward
                state = next_state
            else:
                optdone = False
                opt_func = opt_func_map[action]
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    eps_reward += reward
                    _, optdone = opt_func(env, next_state)
                    state = next_state
        test_rewards.append(eps_reward)
    return test_rewards
```

[ ]:
```
MAINPATH = '/content/drive/MyDrive/Colab Notebooks/CS6700/Assignments/
 ↪Assignment3/Part1'
def plot_reward_curve(eps_reward_list, path, alp, eps, gam=0.9):
    fig, ax = plt.subplots(figsize=(16,8))
    ax.set_title('Reward Curve')
    ax.set_xlabel('Episode Number')
    ax.set_ylabel('Reward')
    num_episodes = len(eps_reward_list)
    ax.plot(np.arange(1, num_episodes+1), eps_reward_list, color='red',␣
 ↪linewidth=0.5)
    props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
    textstr = ' alpha = {}\n epsilon = {}\n gamma = {}'.format(alp,eps,gam)
    ax.text(0.8, 0.2, textstr, transform=ax.transAxes, fontsize=12,␣
 ↪verticalalignment='top', bbox=props)
    fig.savefig(MAINPATH+'/'+path+'/'+'reward_curve.png')
```

[ ]:
```
def visualize_q_values(q_values, msg, path, pass_src = None, pass_dest = None):
    assert(pass_src != None and pass_dest != None)

    req_actions = [[None for _ in range(5)] for _ in range(5)]
    req_q_values = [[None for _ in range(5)] for _ in range(5)]
    temp_env = gym.make('Taxi-v3')
    for s in range(500):
        s_vec = list(temp_env.decode(s))
        if s_vec[2] == pass_src and s_vec[3] == pass_dest:
            req_actions[s_vec[0]][s_vec[1]] = np.argmax(q_values[s])
            req_q_values[s_vec[0]][s_vec[1]] = np.max(q_values[s])
```

```
    req_actions = np.array(req_actions)
    fig, ax = plt.subplots(figsize=(8,8))
    ax.set_title(msg)
    ax.invert_yaxis()
    ax.xaxis.tick_top()
    ax.set_xlabel('Columns')
    ax.xaxis.set_label_position('top')
    ax.set_ylabel('Rows')
    mesh = ax.pcolormesh(req_q_values, edgecolors='k', linewidths=2)
    fig.colorbar(mesh)
    def x_direct(a):
        if a in [4,5,6,7,8]:
            return 0
        if a in [0, 1]:
            return 0
        return 1 if a == 2 else -1
    def y_direct(a):
        if a in [4,5,6,7,8]:
            return 0
        if a in [2, 3]:
            return 0
        return 1 if a == 1 else -1
    idx = np.indices((5,5))
    policyx = np.vectorize(x_direct)(req_actions)
    policyy = np.vectorize(y_direct)(req_actions)
    req_action_dict = {4 : 'Pickup', 5 : 'Drop', 6 : 'R', 7 : 'G', 8 : 'Y', 9 :␣
↪'B'}
    for i,j,px,py in zip(idx[1].ravel(), idx[0].ravel(), policyx.ravel(),␣
↪policyy.ravel()):
        if (req_actions[j, i] < 4):
            ax.quiver(i+0.5, j+0.5, px, py, pivot="middle",␣
↪scale=10,color='red')
        else:
            ax.text(i+0.5, j+0.5, req_action_dict[req_actions[j][i]],␣
↪horizontalalignment='center',verticalalignment='center',color='tomato')
    fig.savefig(MAINPATH+'/'+path+'/'+msg+'.png')
```

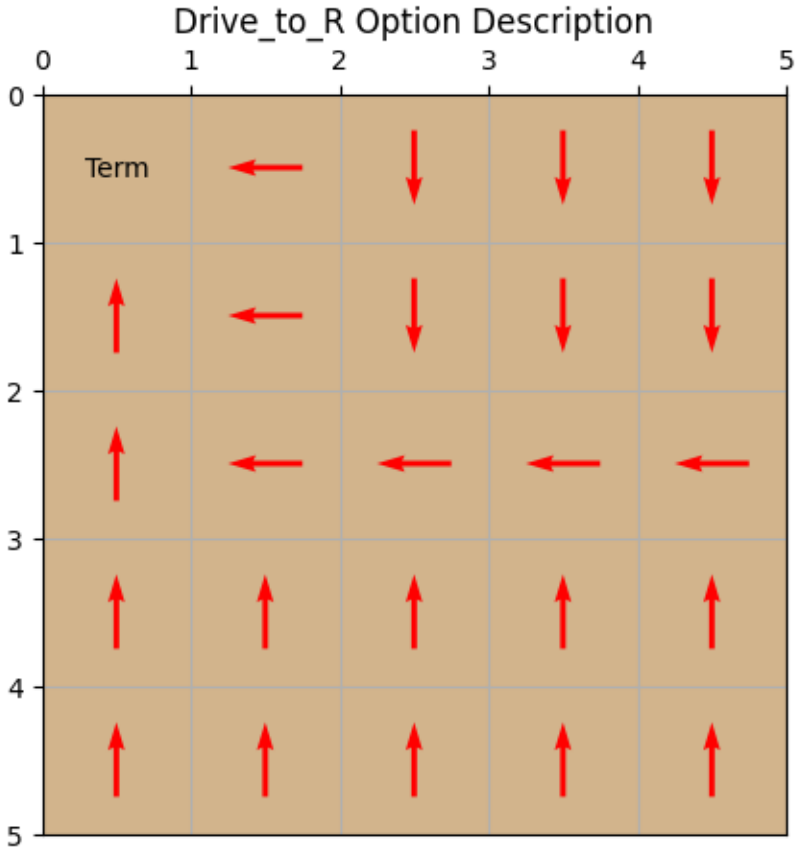
0.4 Defining the DRIVE to {R, G, Y, B} Options

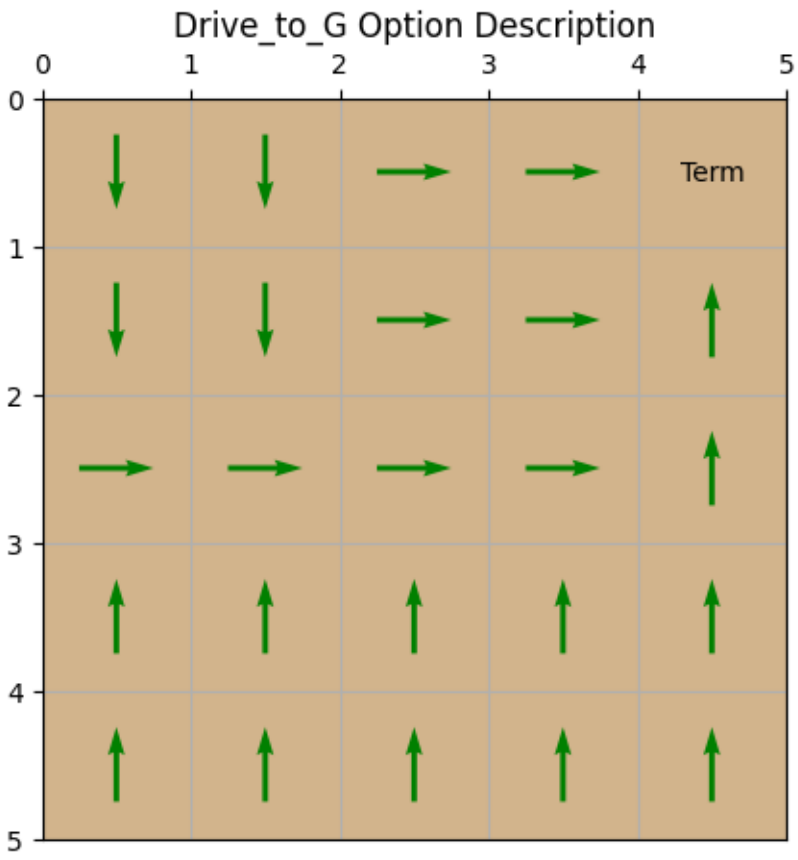
[ ]:
```
PLOTSPATH = MAINPATH + '/OptionDescs'
def save_option_desc(policy, desc, color, end):
    pcopy = np.copy(policy)
    pcopy[end] = -1
    fig, ax = plt.subplots(figsize = (5,5))
    ax.set_title(f'{desc} Option Description')
    ax.set_xlim([0,5])
```

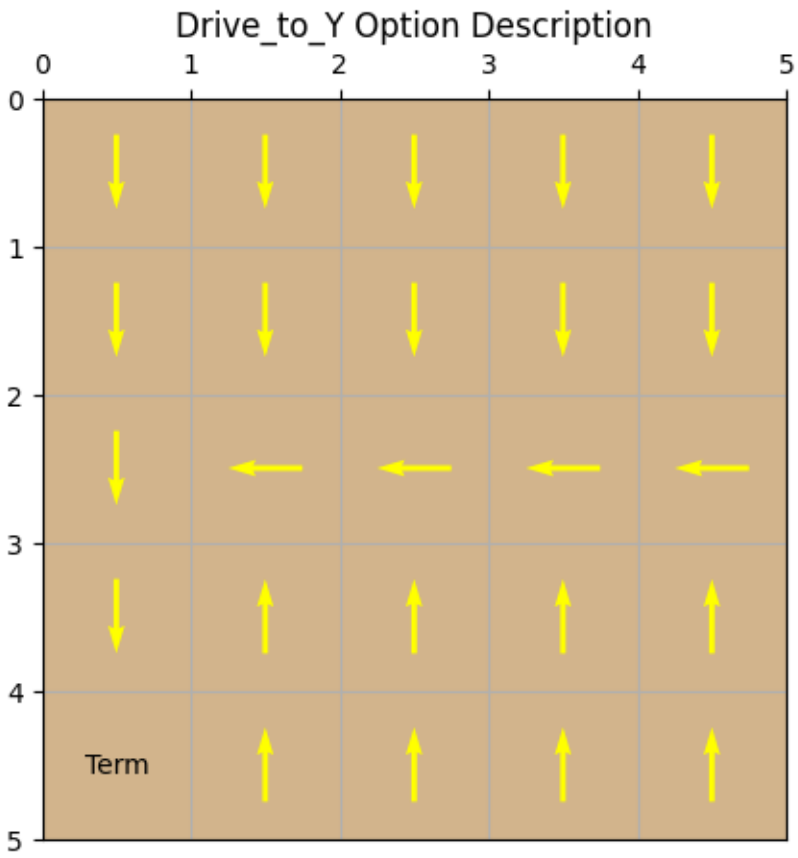
```
    ax.set_ylim([0,5])
    def x_direct(a):
        if a in [-1, 0, 1]:
            return 0
        return 1 if a == 2 else -1
    def y_direct(a):
        if a in [-1, 2, 3]:
            return 0
        return 1 if a == 1 else -1
    policyx = np.vectorize(x_direct)(pcopy)
    policyy = np.vectorize(y_direct)(pcopy)
    idx = np.indices(policy.shape)
    for i,j,px,py in zip(idx[1].ravel(), idx[0].ravel(), policyx.ravel(),␣
 ↪policyy.ravel()):
        if (pcopy[j, i] != -1):
            ax.quiver(i+0.5, j+0.5, px, py, pivot="middle",␣
 ↪scale=10,color=color)
        else:
            ax.text(i+0.5, j+0.5, 'Term',␣
 ↪horizontalalignment='center',verticalalignment='center')
    ax.set_facecolor('tan'); ax.invert_yaxis(); ax.xaxis.tick_top(); ax.grid()
    fig.savefig(f'{PLOTSPATH}/option_{desc}.png')
```

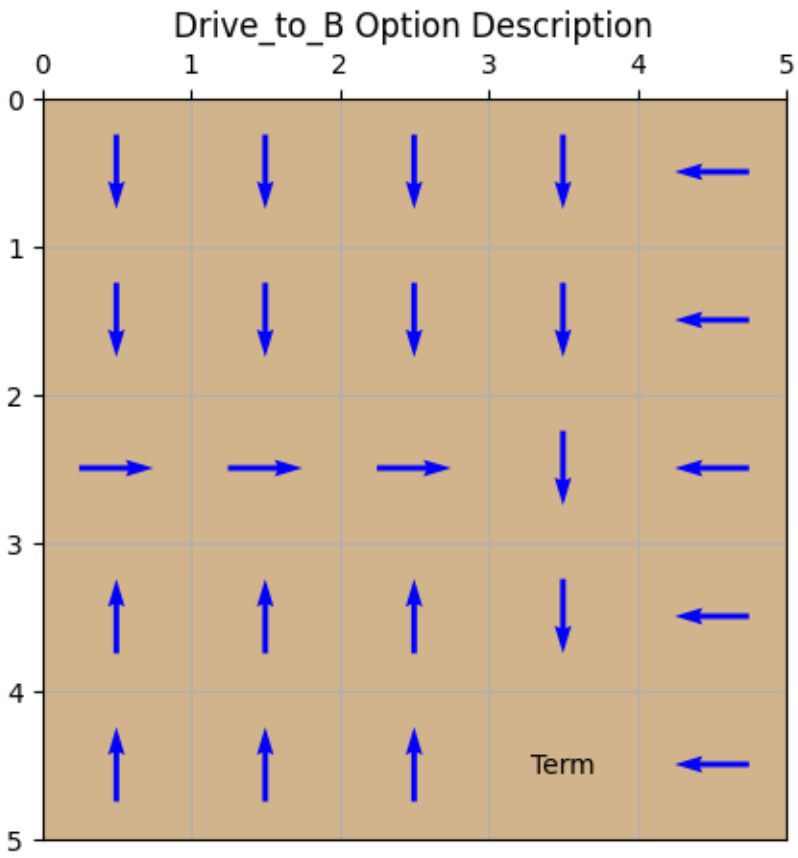
[ ]:
```
# defining the required option action matrix for taking the car to all colors␣
 ↪[FOR EASY CODING]
RED_ACTN_MATRIX = np.array([[1,3,0,0,0],
                            [1,3,0,0,0],
                            [1,3,3,3,3],
                            [1,1,1,1,1],
                            [1,1,1,1,1]])
save_option_desc(RED_ACTN_MATRIX, 'Drive_to_R', "Red", (0,0))
GRE_ACTN_MATRIX = np.array([[0,0,2,2,1],
                            [0,0,2,2,1],
                            [2,2,2,2,1],
                            [1,1,1,1,1],
                            [1,1,1,1,1]])
save_option_desc(GRE_ACTN_MATRIX, 'Drive_to_G', "Green", (0,4))
YEL_ACTN_MATRIX = np.array([[0,0,0,0,0],
                            [0,0,0,0,0],
                            [0,3,3,3,3],
                            [0,1,1,1,1],
                            [0,1,1,1,1]])
save_option_desc(YEL_ACTN_MATRIX, 'Drive_to_Y', "Yellow", (4,0))
BLU_ACTN_MATRIX = np.array([[0,0,0,0,3],
                            [0,0,0,0,3],
                            [2,2,2,0,3],
                            [1,1,1,0,3],
```

```
                              [1,1,1,0,3]])
save_option_desc(BLU_ACTN_MATRIX, 'Drive_to_B', "Blue", (4,3))

RED_TERM_MATRIX = np.zeros((5,5)); RED_TERM_MATRIX[0,0] = 1
GRE_TERM_MATRIX = np.zeros((5,5)); GRE_TERM_MATRIX[0,4] = 1
YEL_TERM_MATRIX = np.zeros((5,5)); YEL_TERM_MATRIX[4,0] = 1
BLU_TERM_MATRIX = np.zeros((5,5)); BLU_TERM_MATRIX[4,3] = 1

OPT_TO_POLICY_MAP = {6 : RED_ACTN_MATRIX, 7 : GRE_ACTN_MATRIX, 8 :␣
 ↪YEL_ACTN_MATRIX, 9 : BLU_ACTN_MATRIX}
OPT_TO_TERM_MAP = {6 : RED_TERM_MATRIX, 7 : GRE_TERM_MATRIX, 8 :␣
 ↪YEL_TERM_MATRIX, 9 : BLU_TERM_MATRIX}
```




Drive_to_G Option Description
0
1
2
3
4
5
Term


Drive_to_Y Option Description
0
1
2
3
4
5
Term



[ ]:
```
# defining the 4 Option functions
def Drive_to_R(env, state):
    coords = list(env.decode(state))[:2]
    optdone = False
    optact = RED_ACTN_MATRIX[coords[0], coords[1]]

    if (coords == [0, 0]): # termination at reaching RED
        optdone = True

    return [optact, optdone]

def Drive_to_G(env, state):
    coords = list(env.decode(state))[:2]
    optdone = False
    optact = GRE_ACTN_MATRIX[coords[0], coords[1]]

    if (coords == [0, 4]): # termination at reaching GREEN
        optdone = True
```

```
    return [optact, optdone]

def Drive_to_Y(env, state):
    coords = list(env.decode(state))[:2]
    optdone = False
    optact = YEL_ACTN_MATRIX[coords[0], coords[1]]

    if (coords == [4, 0]): # termination at reaching YELLOW
        optdone = True

    return [optact, optdone]

def Drive_to_B(env, state):
    coords = list(env.decode(state))[:2]
    optdone = False
    optact = BLU_ACTN_MATRIX[coords[0], coords[1]]

    if (coords == [4, 3]): # termination at reaching BLUE
        optdone = True

    return [optact, optdone]
```

0.5 Exploration Function

[ ]:
```
#Q-Table: (States x Actions) === (env.ns(500) x total actions(10))
q_values_test = np.zeros((500, 10))
uf_values_test = np.zeros((500, 10)) # Update_Frequency Data structure - stores␣
 ↪the number of updates for (s,a) where a = prim action/an option
avl_actions = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # for ["South", "North", "East",␣
 ↪"West", "Pickup", "Drop", "R" opt, "G" opt, "Y" opt, "B" opt]

def egreedy_policy(q_values, state, epsilon, rg, disallow):
    if (rg.random() < epsilon):   # epsilon prob for uniform choice over all␣
 ↪actions and options
        if (disallow != None):
            val_actions = avl_actions[:]; val_actions.remove(disallow)
            return rg.choice(val_actions)
        else:
            return rg.choice(avl_actions)
    else:                         # 1 - epsilon prob for greedy action/option
            return np.argmax(q_values[state])
```

0.6 SMDP Q-Learning

[ ]:
```
#### SMDP Q-Learning

def SMDP_QLearning(alpha, epsilon, epcount = 10000, gamma = 0.9, rg = np.random.
 ↪RandomState(42)):
    env = gym.make('Taxi-v3')
    env.seed(42) # for reproducibility

    # arrays for storing q value, update counts and episode rewards
    q_values = np.zeros((500, 10))
    uf_values = np.zeros((500, 10))
    ep_rewards_list = []
    opt_func_map = {6: Drive_to_R, 7 : Drive_to_G, 8 : Drive_to_Y, 9 :␣
 ↪Drive_to_B} # opt id to option function map
    dis_opts = {(0,0) : 6, (0,4) : 7, (4,0) : 8, (4,3) : 9} # store disallowed␣
 ↪options for each color coordinates in a dict

    # Iterate over epcount episodes
    for epidx in tqdm(range(epcount)):
        state = env.reset()
        done = False
        eps_reward = 0

        # While episode is not over
        while not done:
            # Choose action/option. Note : option should be allowed at the␣
 ↪current co-ordinates
            st_coords = tuple(env.decode(state))[:2]
            dis_opt = (dis_opts[st_coords] if st_coords in dis_opts.keys() else␣
 ↪None)
            action = egreedy_policy(q_values, state, epsilon, rg, dis_opt)
            # Checking if primitive action
            if action < 6:
                # Perform regular Q-Learning update for state-action pair
                next_state, reward, done, _ = env.step(action)
                q_values[state, action] += alpha * (reward + gamma*np.
 ↪max(q_values[next_state]) - q_values[state, action])
                uf_values[state,action] += 1
                state = next_state
                eps_reward += reward
            else:
                # find the corresponding function for the option
                opt_func = opt_func_map[action]
                reward_bar = 0

                initial_state = state
```

```
                opt_steps = 0
                optdone = False
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    _, optdone = opt_func(env, next_state)
                    reward_bar = gamma * reward_bar + reward
                    opt_steps += 1
                    state = next_state
                    eps_reward += reward

                # SMDP Q-Learning Update for options. We update the q-value of␣
 ↪the (initial_state, option) pair at the end of option execution.
                q_values[initial_state, action] += alpha * (reward_bar + (gamma␣
 ↪** opt_steps) * np.max(q_values[state]) - q_values[initial_state, action])
                uf_values[initial_state,action] += 1
        ep_rewards_list.append(eps_reward)
    return q_values, uf_values, ep_rewards_list
```

0.7 Intra Option Q-Learning

[ ]:
```
#### Intra Option Q-Learning

# NOT USED - INSIGNIFICANT SPEED BOOST
def gen_policies_sharing_action():
    ans = [[[[] for _ in range(6)] for _ in range(5)] for _ in range(5)]
    for opt in [6, 7, 8, 9]:
        policy = OPT_TO_POLICY_MAP[opt]
        for i in range(5):
            for j in range(5):
                ans[i][j][policy[i,j]].append(opt)
    return ans

def IntraOption_QLearning(alpha, epsilon, epcount = 10000, gamma = 0.9, rg = np.
 ↪random.RandomState(42)):
    env = gym.make('Taxi-v3')
    env.seed(42) # for reproducibility

    # arrays for storing q value, update counts and episode rewards
    q_values = np.zeros((500, 10))
    uf_values = np.zeros((500, 10))
    ep_rewards_list = []
    opt_func_map = {6: Drive_to_R, 7 : Drive_to_G, 8 : Drive_to_Y, 9 :␣
 ↪Drive_to_B} # opt id to option function map
    dis_opts = {(0,0) : 6, (0,4) : 7, (4,0) : 8, (4,3) : 9} # store disallowed␣
 ↪options for each color coordinates in a dict
```

```
    # Iterate over epcount episodes
    for epidx in tqdm(range(epcount)):
        state = env.reset()
        done = False
        eps_reward = 0

        # While episode is not over
        while not done:
            # Choose action/option. Note : option should be allowed at the␣
↪current co-ordinates
            st_coords = tuple(env.decode(state))[:2]
            dis_opt = (dis_opts[st_coords] if st_coords in dis_opts.keys() else␣
↪None)
            action = egreedy_policy(q_values, state, epsilon, rg, dis_opt)

            # Checking if primitive action
            if action < 6:
                # Perform regular Q-Learning update for state-action pair
                next_state, reward, done, _ = env.step(action)
                q_values[state, action] += alpha * (reward + gamma*np.
↪max(q_values[next_state]) - q_values[state, action])
                uf_values[state,action] += 1
                state = next_state
                eps_reward += reward
            else:
                # find the corresponding function for the option
                opt_func = opt_func_map[action]
                other_opts = [6,7,8,9]; other_opts.remove(action)
                optdone = False
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    _, optdone = opt_func(env, next_state)
                    eps_reward += reward
                    # perform update for the optact - primitive action
                    q_values[state, optact] += alpha * (reward + gamma*np.
↪max(q_values[next_state]) - q_values[state, optact])
                    uf_values[state, optact] += 1

                    # find all options that choose the same primitive action at␣
↪state
                    opt_upd_list = [action]
                    st_coords = list(env.decode(state))[:2]
                    for oth in other_opts:
                        if OPT_TO_POLICY_MAP[oth][st_coords[0], st_coords[1]]␣
↪== optact:
```

```
                            opt_upd_list.append(oth)
                    # debug - print(st_coords, action, opt_upd_list)
                    # perform updates for all options that choose the same␣
↪primitive action at state
                    nst_coords = list(env.decode(next_state))[:2]
                    for opt in opt_upd_list:
                        term_matrix = OPT_TO_TERM_MAP[opt]
                        if term_matrix[nst_coords[0], nst_coords[1]] == 1:
                            # if the option terminates, we do total max over␣
↪all actions(and options) in next state
                            q_values[state, opt] += alpha * (reward + gamma *␣
↪np.max(q_values[next_state]) - q_values[state, opt])
                            uf_values[state, opt] += 1
                        else:
                            # if it does not, we use the option q-value in next␣
↪state for update
                            q_values[state, opt] += alpha * (reward + gamma *␣
↪(q_values[next_state, opt]) - q_values[state, opt])
                            uf_values[state, opt] += 1

                    state = next_state
        ep_rewards_list.append(eps_reward)
    return q_values, uf_values, ep_rewards_list
```

0.8 Wandb Sweeping For finding best hyperparameters

[ ]:
```
!pip install wandb
import wandb
wandb.login()
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting wandb
  Downloading wandb-0.14.2-py3-none-any.whl (2.0 MB)
                            2.0/2.0 MB
38.8 MB/s eta 0:00:00
Collecting GitPython!=3.1.29,>=1.0.0
  Downloading GitPython-3.1.31-py3-none-any.whl (184 kB)
                           184.3/184.3 kB
21.5 MB/s eta 0:00:00
Collecting docker-pycreds>=0.4.0
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.15.0 in
/usr/local/lib/python3.9/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: Click!=8.0.0,>=7.0 in
/usr/local/lib/python3.9/dist-packages (from wandb) (8.1.3)
Requirement already satisfied: typing-extensions in
```

```
/usr/local/lib/python3.9/dist-packages (from wandb) (4.5.0)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.9/dist-
packages (from wandb) (5.9.4)
Requirement already satisfied: appdirs>=1.4.3 in /usr/local/lib/python3.9/dist-
packages (from wandb) (1.4.4)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.9/dist-packages
(from wandb) (6.0)
Collecting sentry-sdk>=1.0.0
  Downloading sentry_sdk-1.19.1-py2.py3-none-any.whl (199 kB)
                           199.2/199.2 kB
18.1 MB/s eta 0:00:00
Requirement already satisfied: setuptools in
/usr/local/lib/python3.9/dist-packages (from wandb) (67.6.1)
Collecting pathtools
  Downloading pathtools-0.1.2.tar.gz (11 kB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: requests<3,>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from wandb) (2.27.1)
Collecting setproctitle
  Downloading setproctitle-1.3.2-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_6
4.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (30 kB)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.9/dist-
packages (from docker-pycreds>=0.4.0->wandb) (1.16.0)
Collecting gitdb<5,>=4.0.1
  Downloading gitdb-4.0.10-py3-none-any.whl (62 kB)
                            62.7/62.7 kB
8.5 MB/s eta 0:00:00
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.0.0->wandb)
(1.26.15)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-
packages (from requests<3,>=2.0.0->wandb) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.0.0->wandb)
(2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.0.0->wandb) (2.0.12)
Collecting smmap<6,>=3.0.1
  Downloading smmap-5.0.0-py3-none-any.whl (24 kB)
Building wheels for collected packages: pathtools
  Building wheel for pathtools (setup.py) … done
  Created wheel for pathtools: filename=pathtools-0.1.2-py3-none-any.whl
size=8807
sha256=285162230bd1f4b09383a585a3527d27b4b27a54d3b2cf3bea6111c8daedd447
  Stored in directory: /root/.cache/pip/wheels/b7/0a/67/ada2a22079218c75a88361c0
782855cc72aebc4d18d0289d05
Successfully built pathtools
Installing collected packages: pathtools, smmap, setproctitle, sentry-sdk,
```

```
docker-pycreds, gitdb, GitPython, wandb
Successfully installed GitPython-3.1.31 docker-pycreds-0.4.0 gitdb-4.0.10
pathtools-0.1.2 sentry-sdk-1.19.1 setproctitle-1.3.2 smmap-5.0.0 wandb-0.14.2
```

```
<IPython.core.display.Javascript object>
```

```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server
locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here:
https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to
quit:
 ..........
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
```

[ ]:
```
True
```

0.8.1 SMDP Q-Learning Hyperparameter Finetuning

[ ]:
```
def test_smdp():
    run = wandb.init(reinit=True)
    al = wandb.config.alpha
    epsn = wandb.config.epsilon
    epcount = wandb.config.train_episodes
    run.name = 'al={:.4f}_eps={:.4f}'.format(al, epsn)
    q_values, uf_values, ep_reward_list = SMDP_QLearning(al, epsn, epcount)

    for i, rew in enumerate(ep_reward_list):
        wandb.log({'train episode number': i+1, 'train_reward' : rew})

    test_rewards = test_agent(q_values, wandb.config.test_episodes)
    for i, rew in enumerate(test_rewards):
        wandb.log({'test episode number': i+1, 'test_reward' : rew})

    wandb.log({'avg_test_reward' : np.mean(test_rewards)})
    run.finish()
```

[ ]:
```
sweep_config = {
    'method' : 'bayes',
    'name' : 'alpha_epsilon_sweep',
    'metric' : {
        'goal' : 'maximize',
        'name' : 'avg_test_reward'
    },
    'parameters' : {
        'alpha' : {'min' : 0.2, 'max' : 0.7},
```

```
        'epsilon' : {'min' : 0.02, 'max' : 0.25},
        'train_episodes' : {'value' : 20000},
        'test_episodes' : {'value' : 1000}
    }
}

sweep_id = wandb.sweep(sweep=sweep_config, project='RL-A3-P1-SMDP',␣
 ↪entity='cs6700_team_2023')

wandb.agent(sweep_id, test_smdp, 'cs6700_team_2023', count=100)
```

0.8.2 Intra-Option Q-Learning Hyperparameter Finetuning

[ ]:
```
def test_intra():
    run = wandb.init(reinit=True)
    al = wandb.config.alpha
    epsn = wandb.config.epsilon
    epcount = wandb.config.train_episodes
    run.name = 'al={:.4f}_eps={:.4f}'.format(al, epsn)
    q_values, uf_values, ep_reward_list = IntraOption_QLearning(al, epsn,␣
 ↪epcount)

    for i, rew in enumerate(ep_reward_list):
        wandb.log({'train episode number': i+1, 'train_reward' : rew})

    test_rewards = test_agent(q_values, wandb.config.test_episodes)
    for i, rew in enumerate(test_rewards):
        wandb.log({'test episode number': i+1, 'test_reward' : rew})

    wandb.log({'avg_test_reward' : np.mean(test_rewards)})
    run.finish()
```

[ ]:
```
sweep_config = {
    'method' : 'bayes',
    'name' : 'alpha_epsilon_sweep',
    'metric' : {
        'goal' : 'maximize',
        'name' : 'avg_test_reward'
    },
    'parameters' : {
        'alpha' : {'min' : 0.2, 'max' : 0.7},
        'epsilon' : {'min' : 0.02, 'max' : 0.25},
        'train_episodes' : {'value' : 20000},
        'test_episodes' : {'value' : 1000}
    }
}
```

```
sweep_id = wandb.sweep(sweep=sweep_config,␣
↪project='RL-A3-P1-IntraOptionQLearning', entity='cs6700_team_2023')
wandb.agent(sweep_id, test_intra, 'cs6700_team_2023', count=100)
```

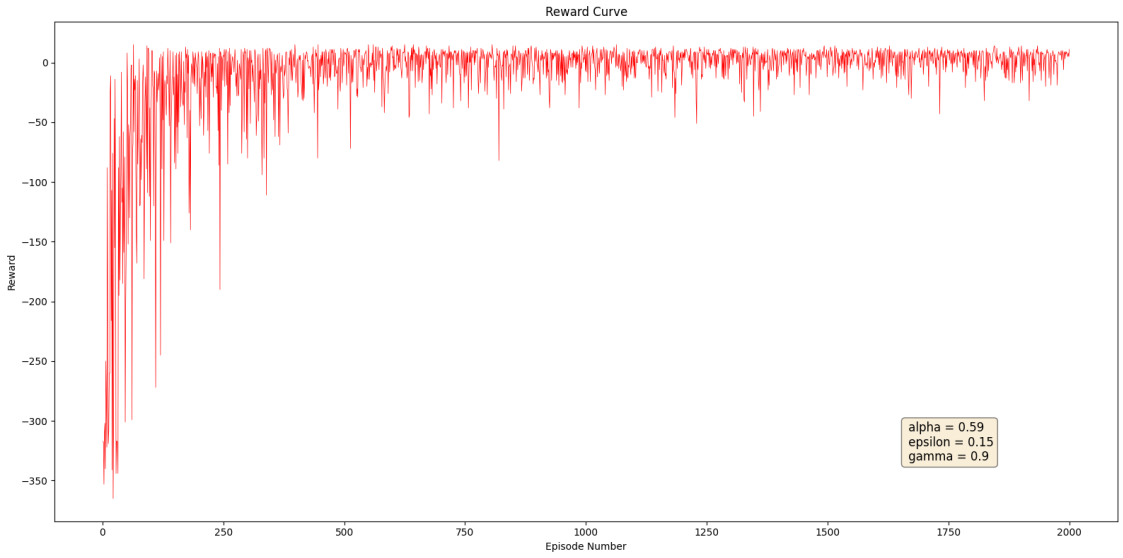
0.9 Generating Plots for Best Hyperparameter Combination

0.9.1 SMDP Q-Learning

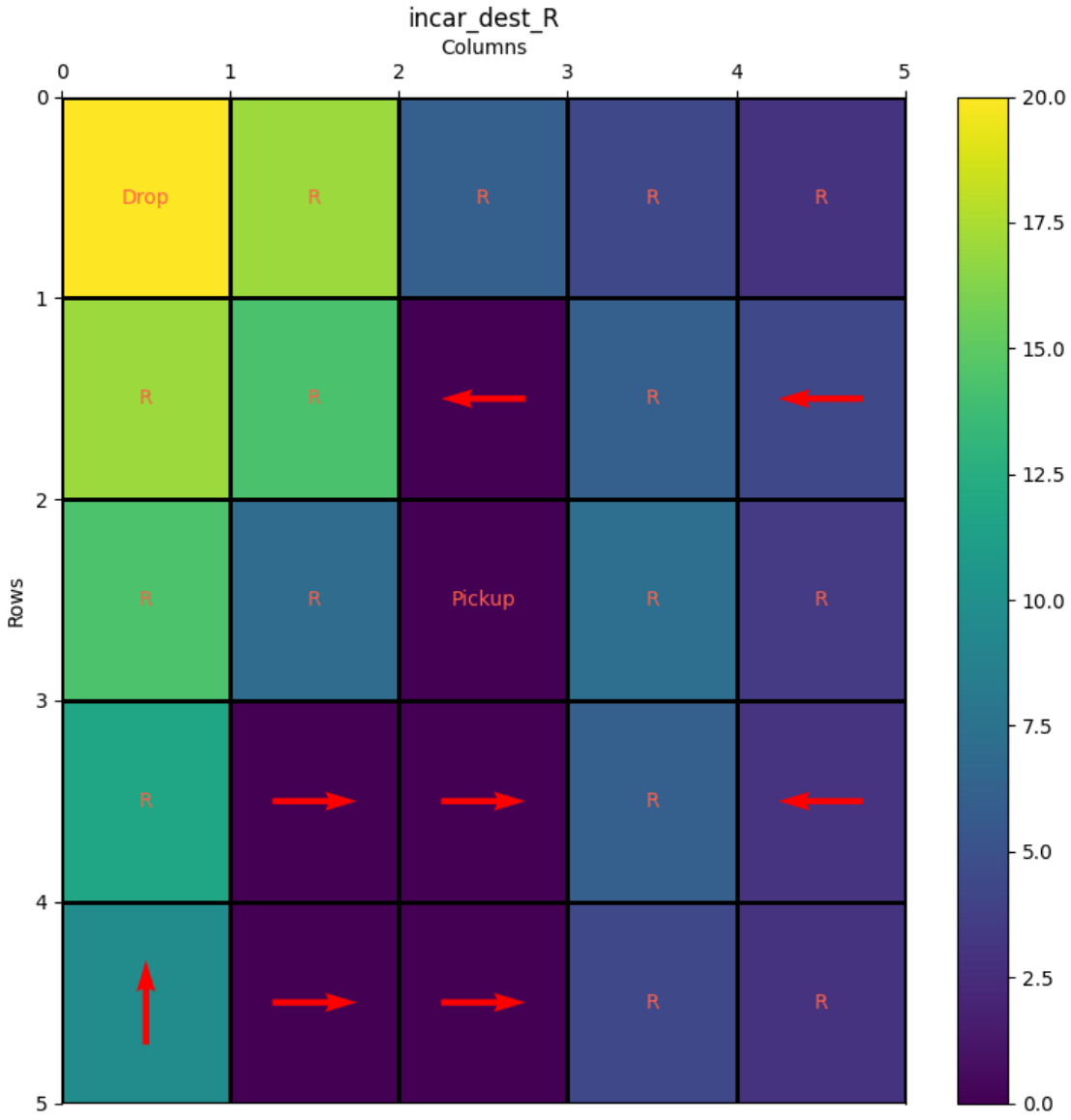
[ ]:
```
bestalpha = 0.59
bestepsilon = 0.15

q_values_smdp, uf_values_smdp, eps_rewards_smdp = SMDP_QLearning(bestalpha,␣
↪bestepsilon, 50000) # 30k to 50k
```

```
100%|      | 50000/50000 [00:34<00:00, 1428.81it/s]
```

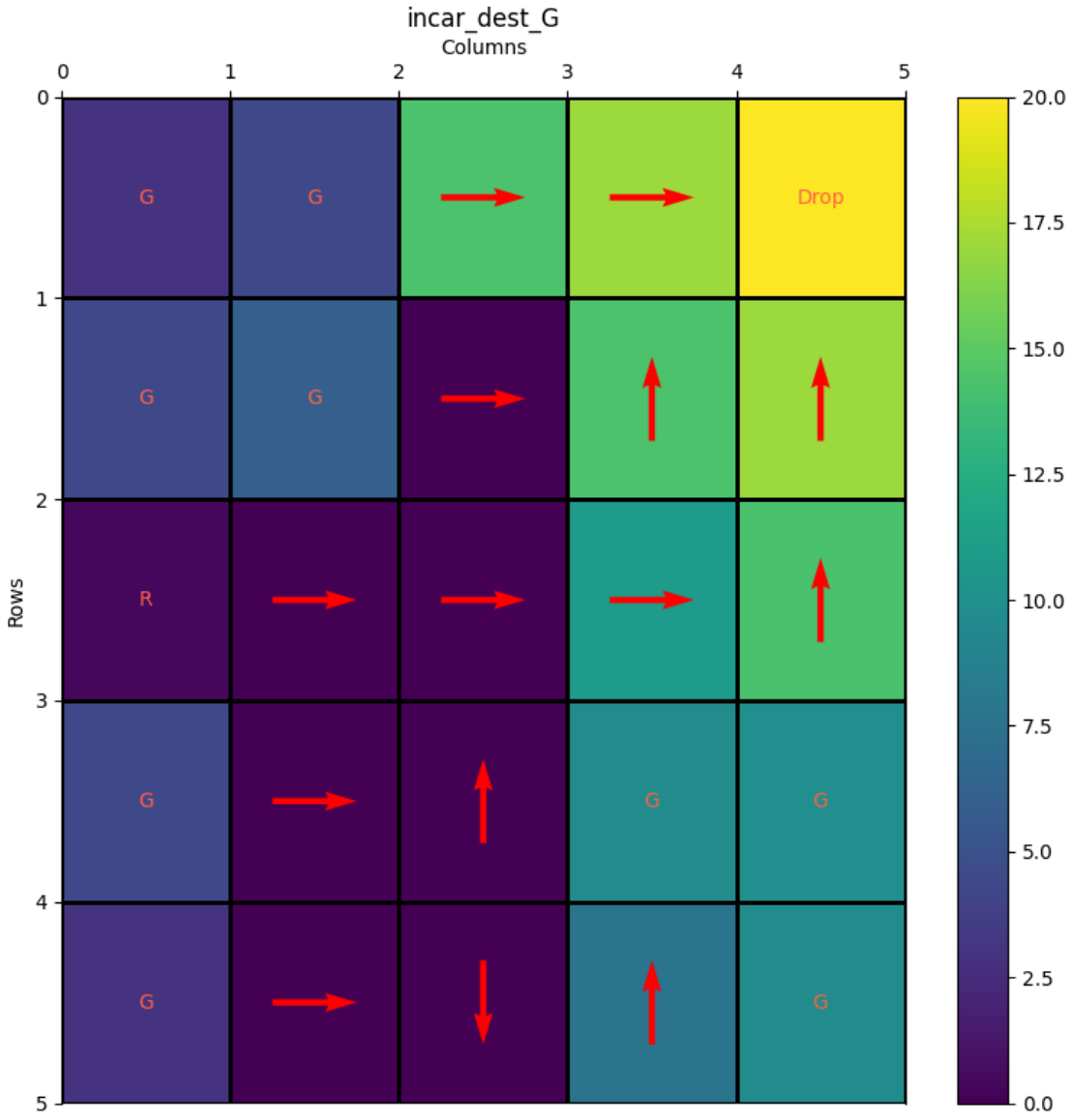
[ ]:
```
plot_reward_curve(eps_rewards_smdp[:2000], 'SMDP', bestalpha, bestepsilon) # 2k␣
↪to 3k
```



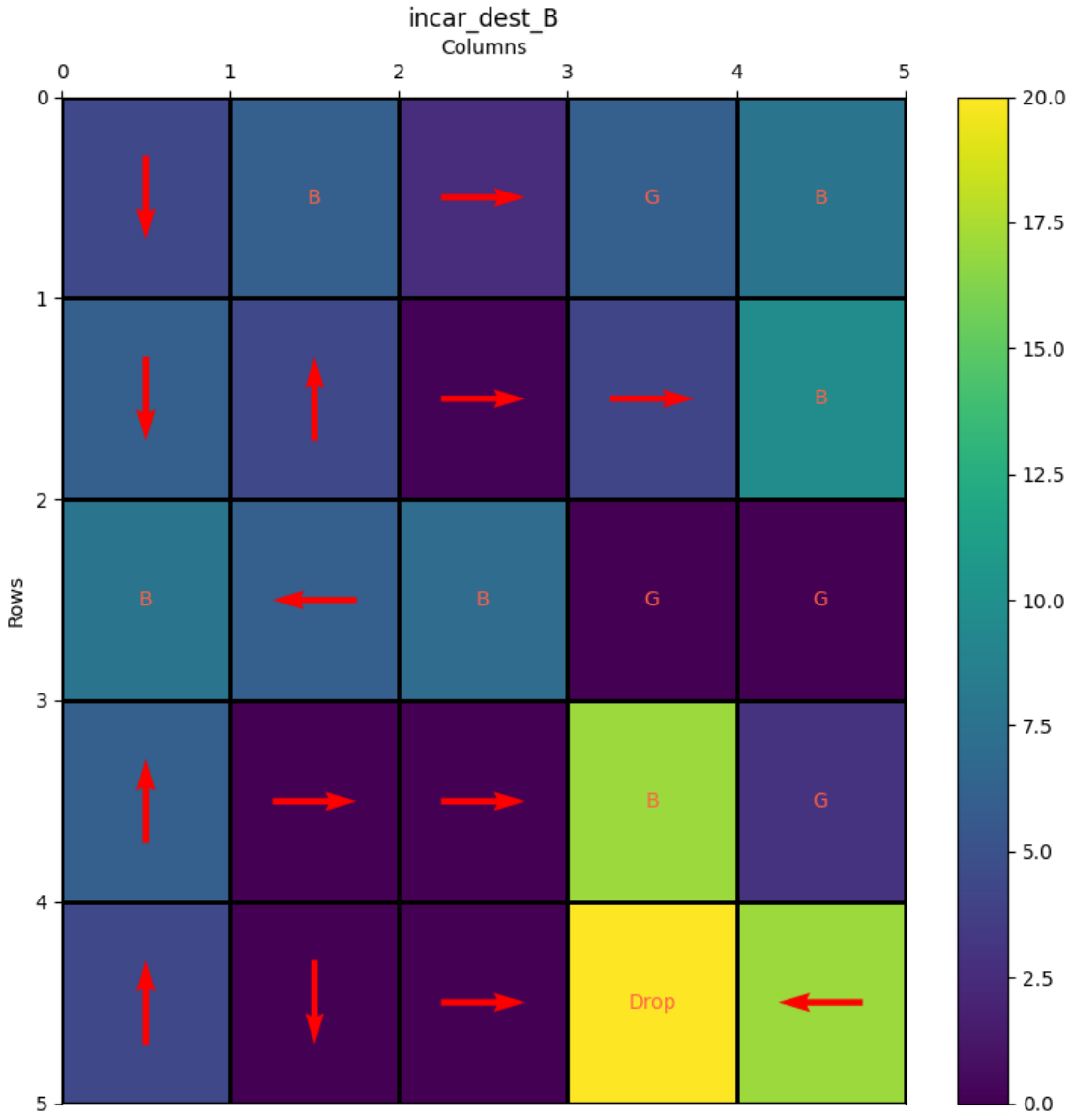
[ ]:
```
visualize_q_values(q_values_smdp, "incar_dest_R", 'SMDP', 4, 0)
```



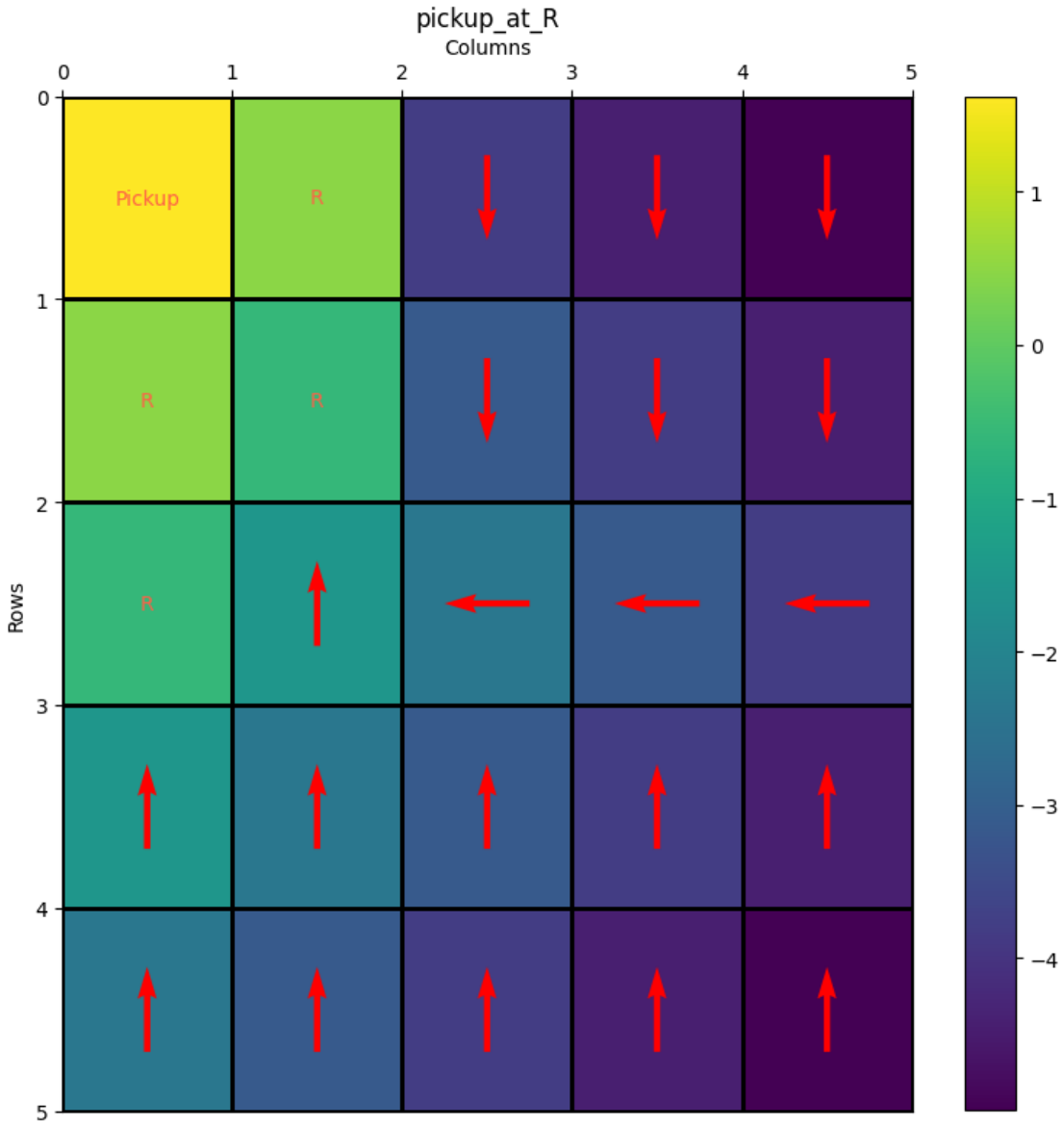
[ ]:
```
visualize_q_values(q_values_smdp, "incar_dest_G", 'SMDP', 4, 1)
```



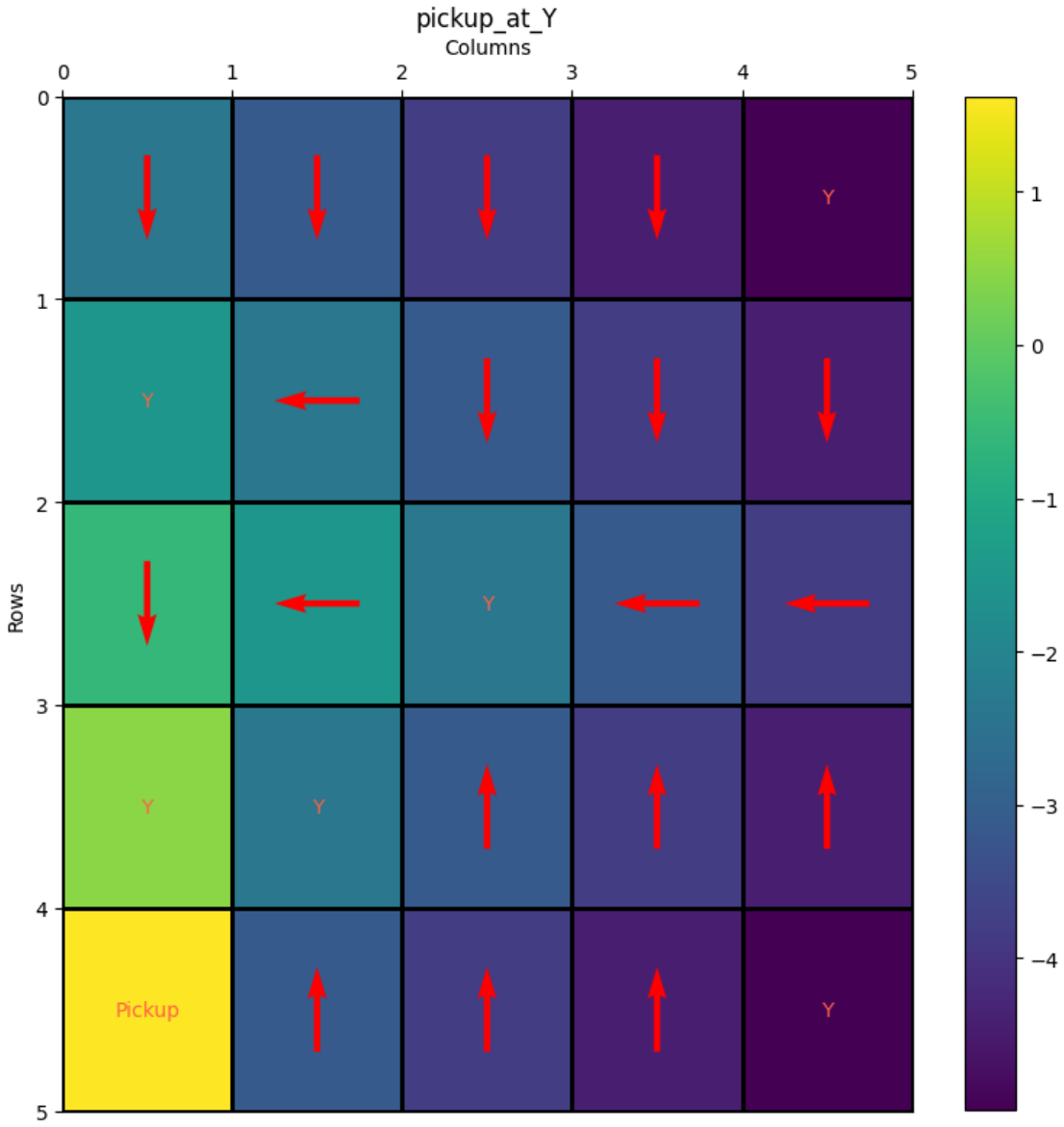
[ ]:
```
visualize_q_values(q_values_smdp, "incar_dest_B", 'SMDP', 4, 3)
```



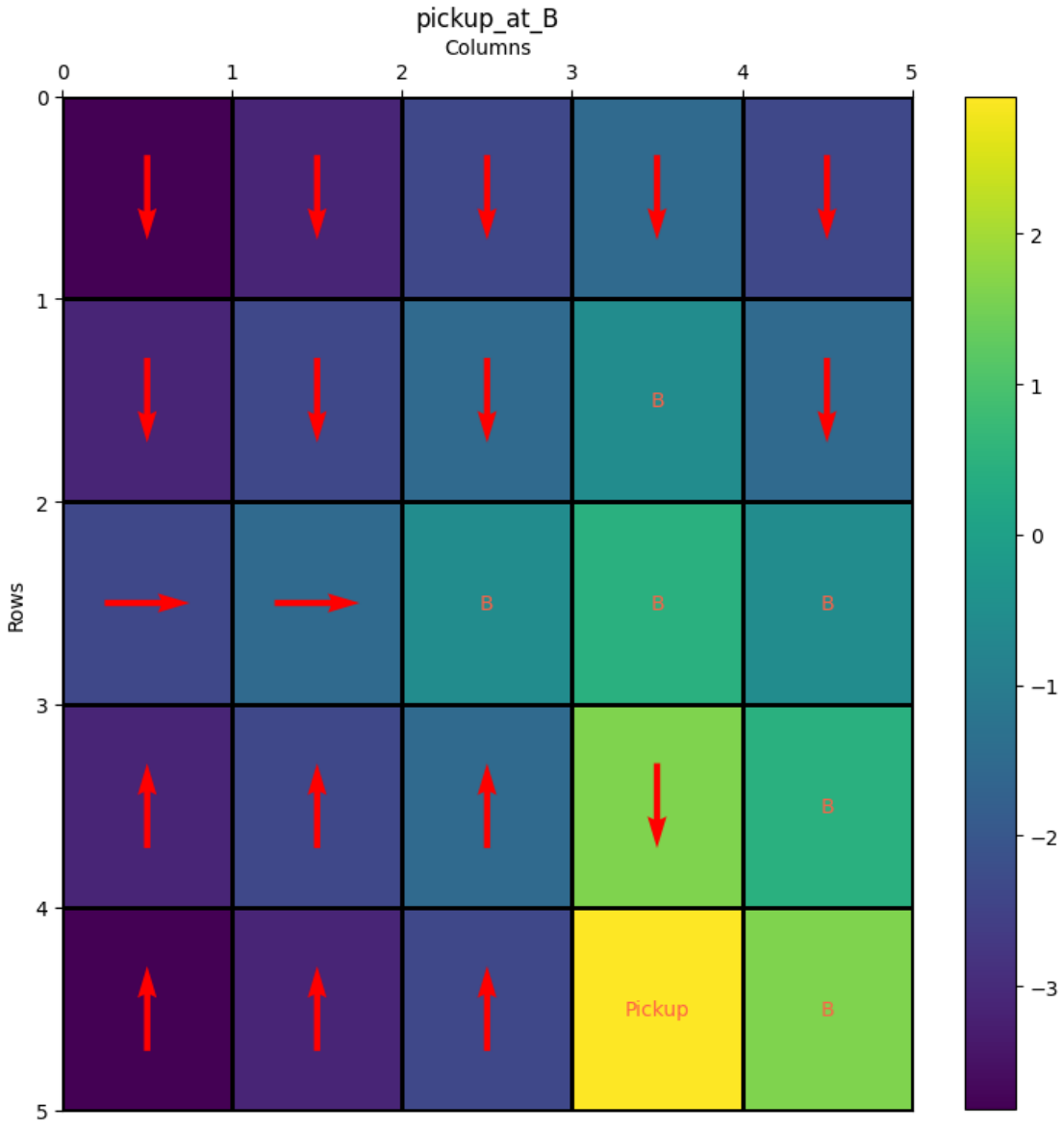
[ ]:
```
visualize_q_values(q_values_smdp, "pickup_at_R", 'SMDP', 0, 1)
```



[ ]:
```
visualize_q_values(q_values_smdp, "pickup_at_Y", 'SMDP', 2, 1)
```



[ ]:
```
visualize_q_values(q_values_smdp, "pickup_at_B", 'SMDP', 3, 2)
```

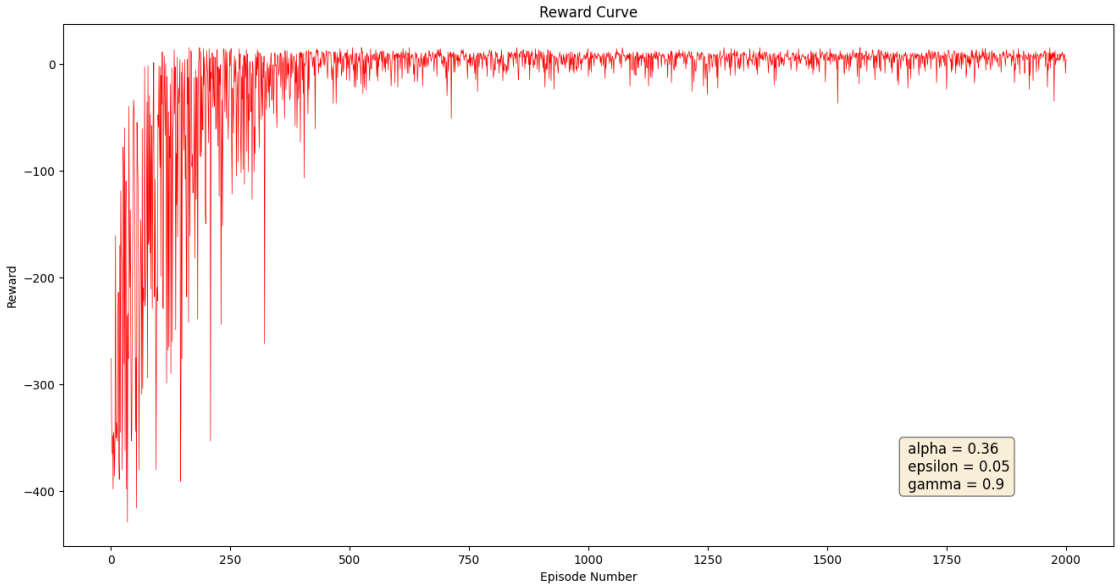


0.9.2 Intra-Option Q-Learning

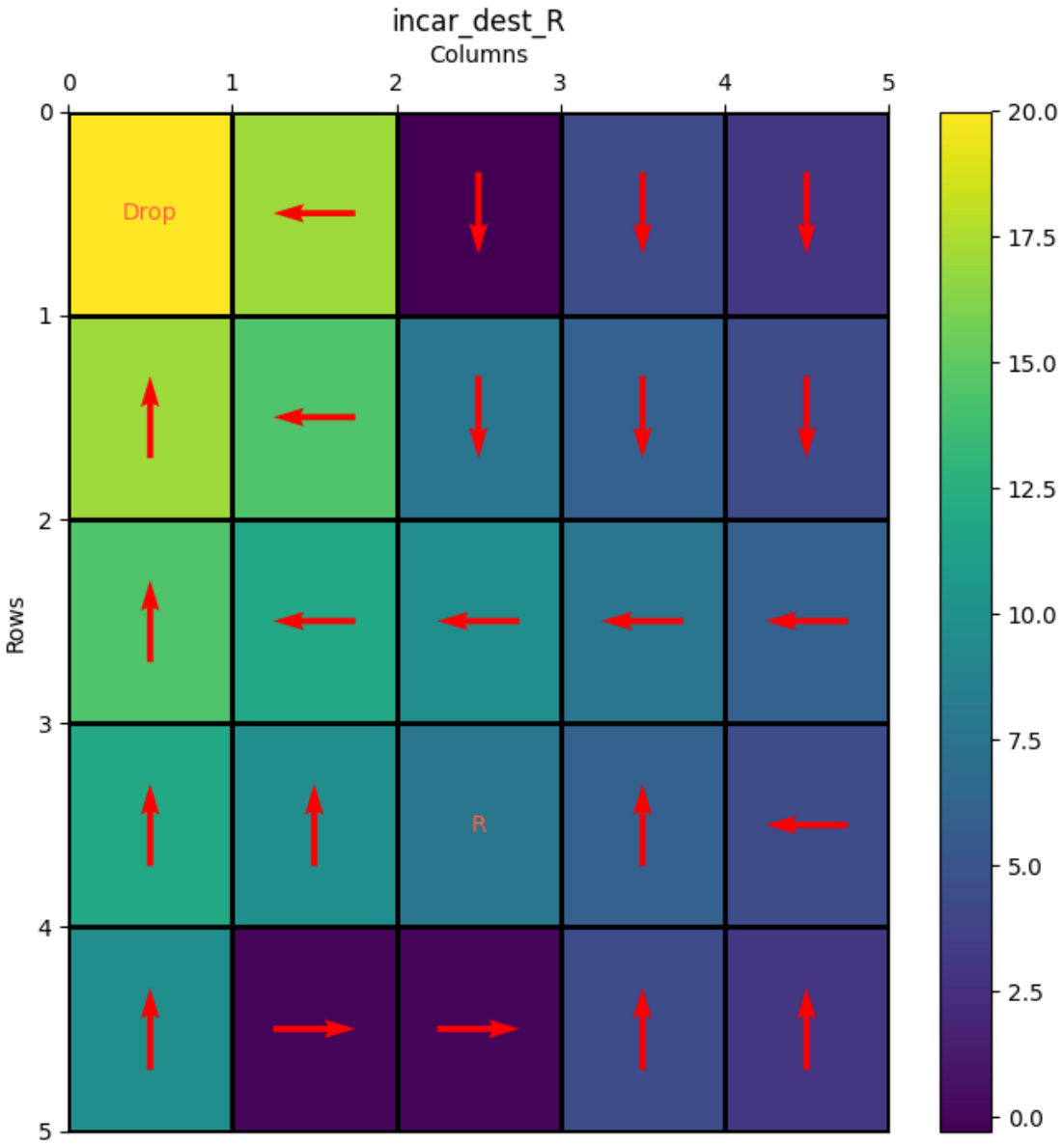
[ ]:
```
bestalpha = 0.36
bestepsilon = 0.05

q_values_intra, uf_values_intra, eps_rewards_intra =␣
↪IntraOption_QLearning(bestalpha, bestepsilon, 50000) # 30k to 50k
```

```
100%|      | 50000/50000 [01:08<00:00, 731.27it/s]
```

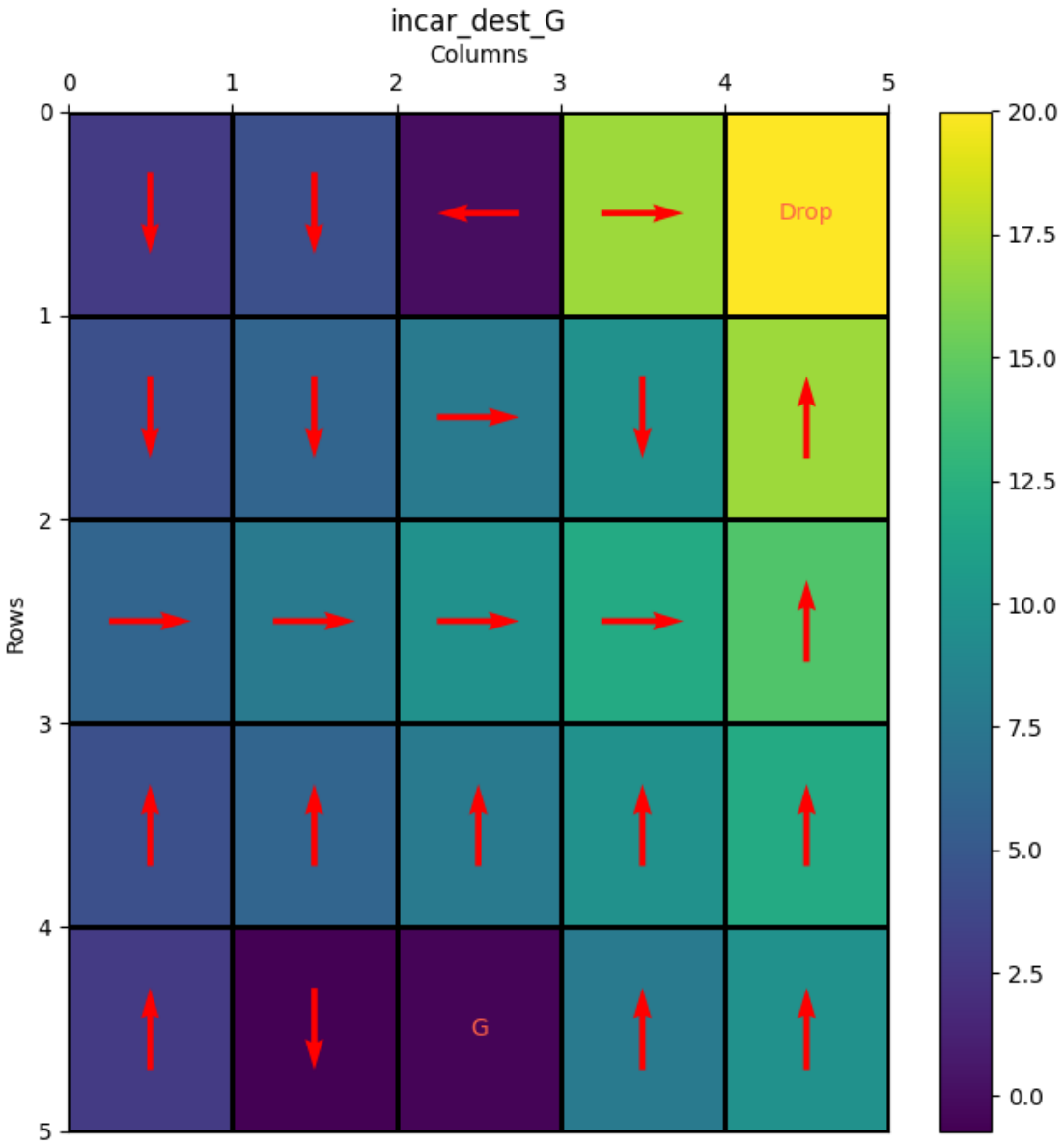
[ ]:
```
plot_reward_curve(eps_rewards_intra[:2000], 'Intra', bestalpha, bestepsilon) #␣
↪2k to 3k
```



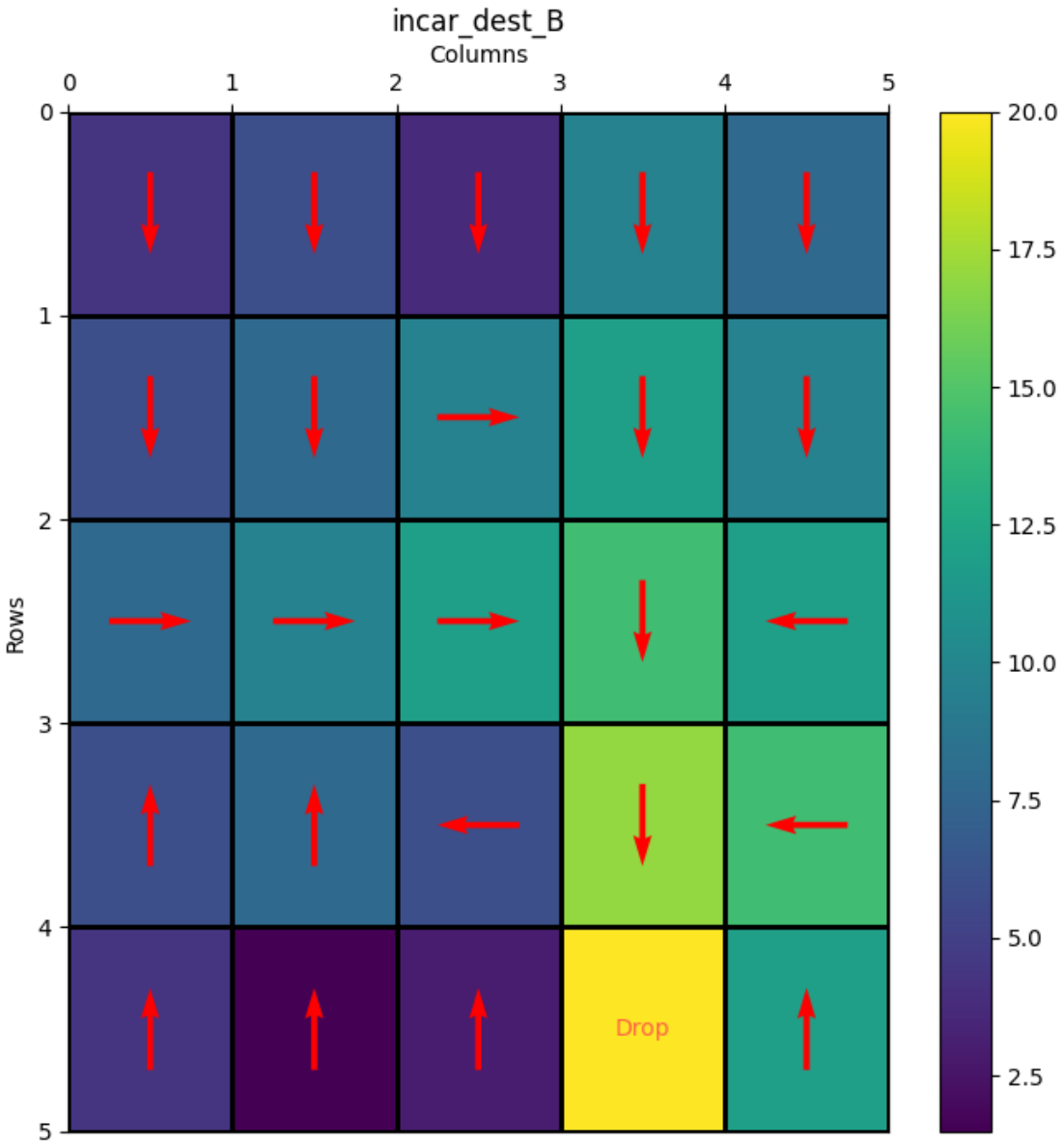
[ ]:
```
visualize_q_values(q_values_intra, "incar_dest_R", 'Intra', 4, 0)
```



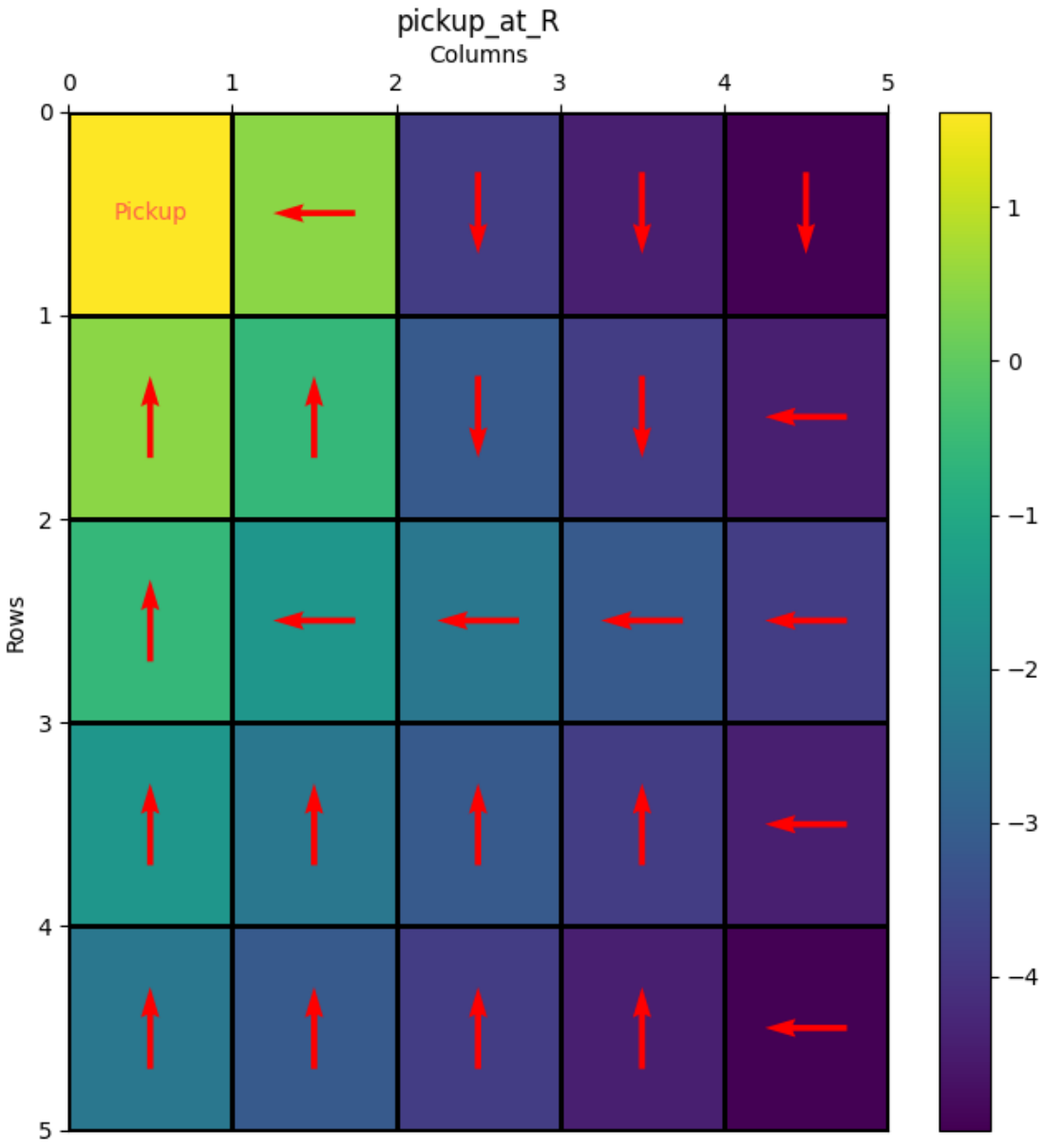
[ ]:
```
visualize_q_values(q_values_intra, "incar_dest_G", 'Intra', 4, 1)
```



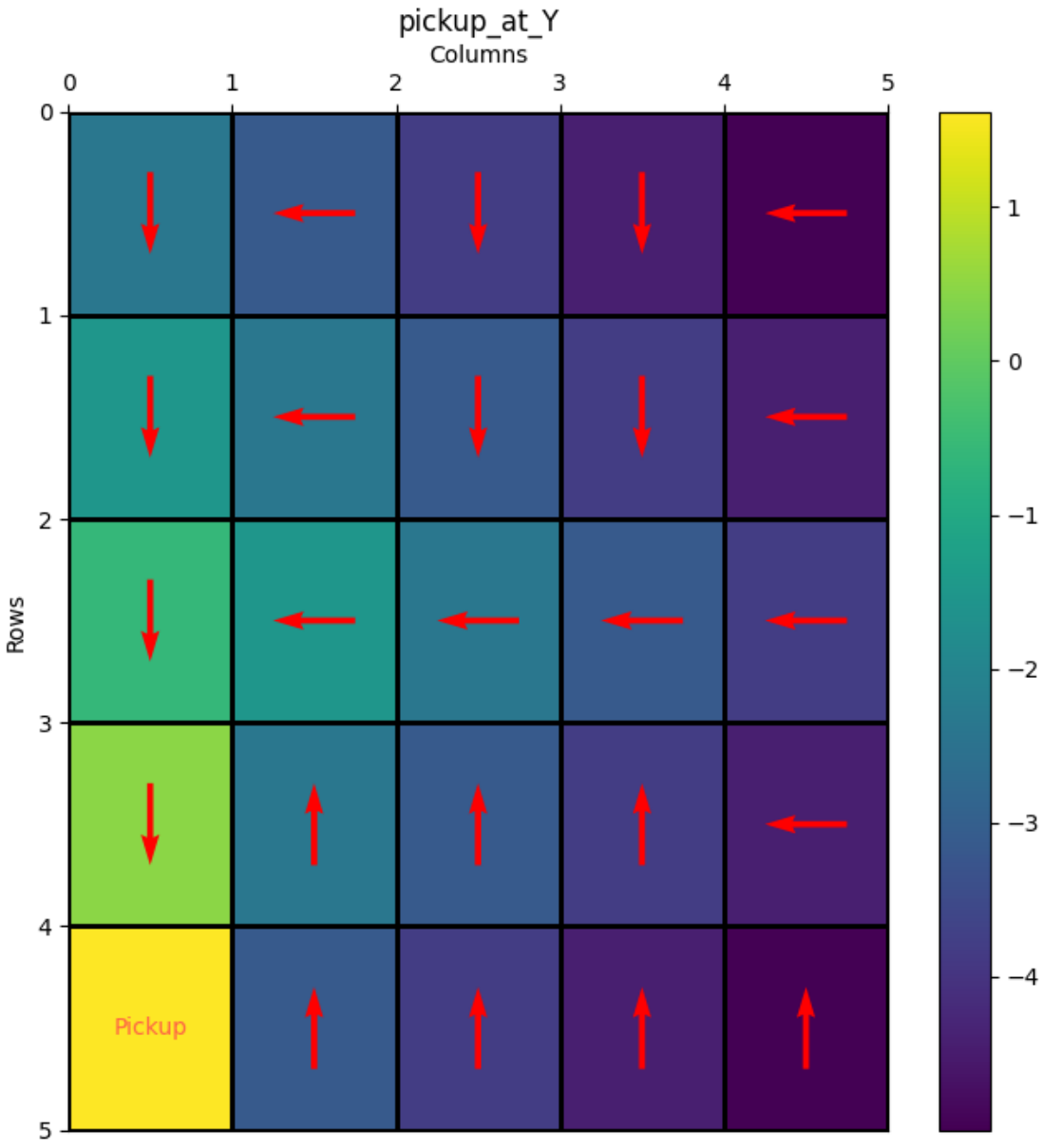

```
[ ]: visualize_q_values(q_values_intra, "incar_dest_B", 'Intra', 4, 3)
```



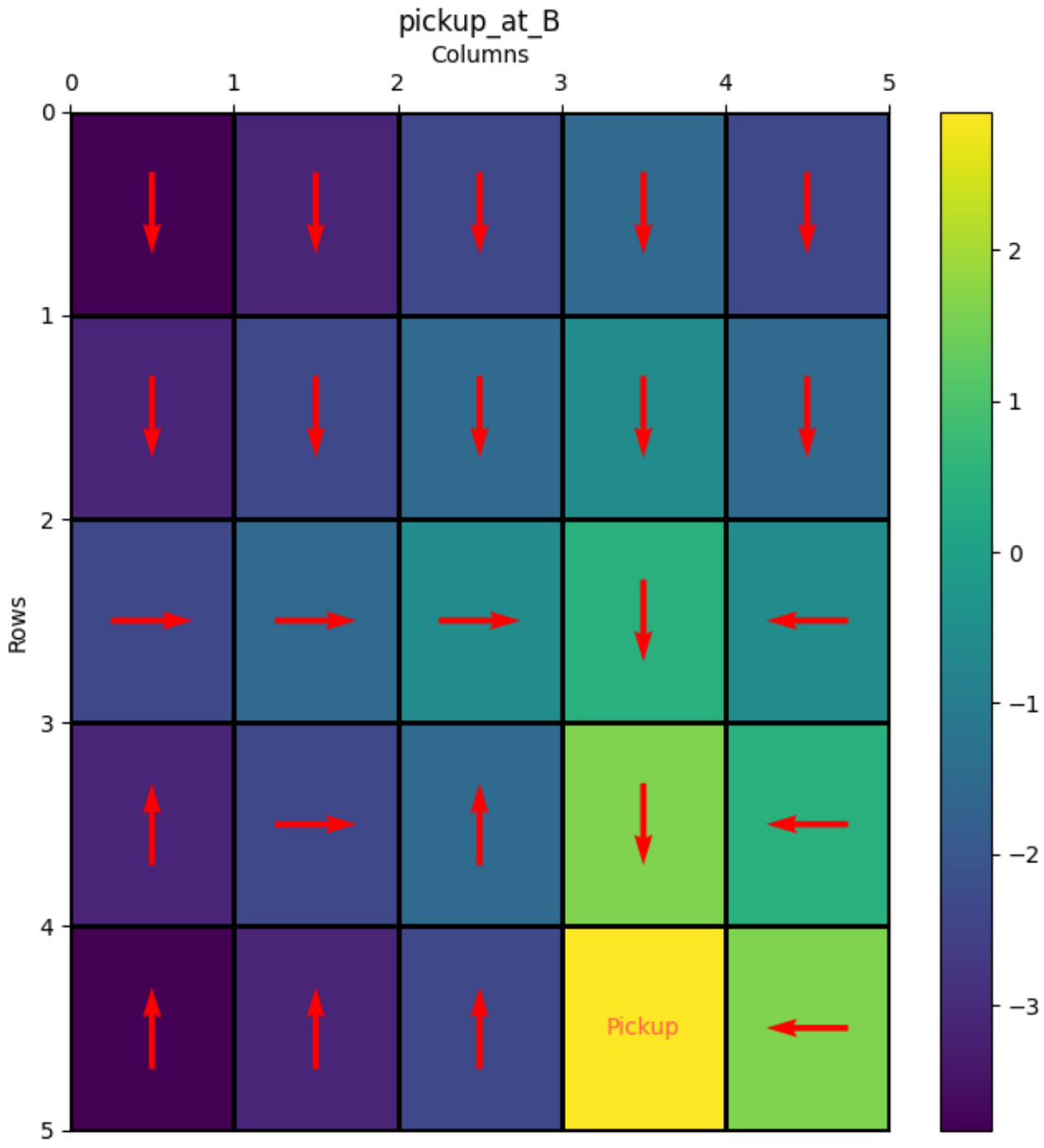
[ ]:
```
visualize_q_values(q_values_intra, "pickup_at_R", 'Intra', 0, 1)
```



[ ]:
```
visualize_q_values(q_values_intra, "pickup_at_Y", 'Intra', 2, 1)
```



[ ]:
```
visualize_q_values(q_values_intra, "pickup_at_B", 'Intra', 3, 2)
```



1 Comparing SMDP and Intra-option Q learning:

[ ]:
```
bestalpha = 0.4
bestepsilon = 0.15

q_values_smdp, uf_values_smdp, eps_rewards_smdp = SMDP_QLearning(bestalpha,␣
↪bestepsilon, 50000) # 30k to 50k
```

```
100%|      | 50000/50000 [00:50<00:00, 989.79it/s]
```

[ ]:
```
bestalpha = 0.4
bestepsilon = 0.15

q_values_intra, uf_values_intra, eps_rewards_intra =␣
 ↪IntraOption_QLearning(bestalpha, bestepsilon, 50000) # 30k to 50k
```

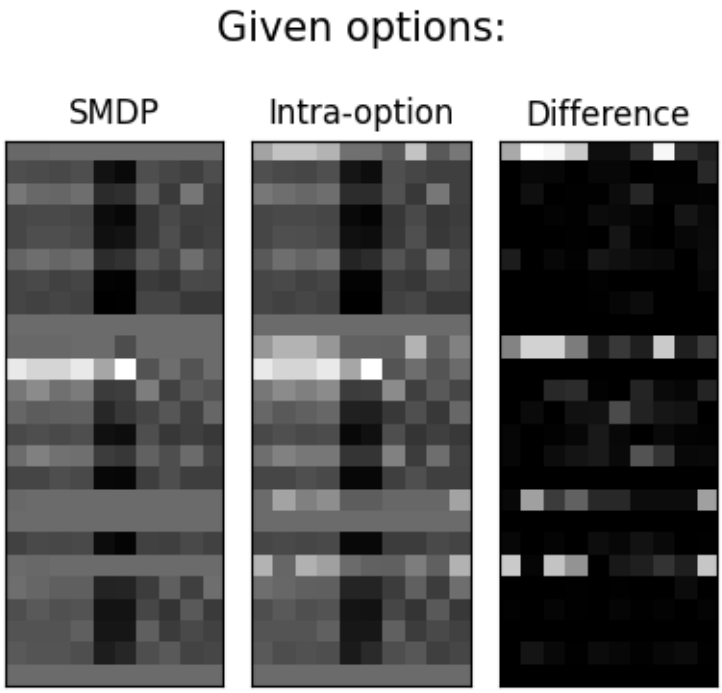
```
100%|      | 50000/50000 [01:00<00:00, 821.89it/s]
```

[ ]:
```
def plotter(image, transf_image,diff, title_1, title_2):
    fig, (ax1, ax2,ax3) = plt.subplots(1, 3, figsize=(4,4))
    fig. suptitle("Given options:", fontsize=15)
    ax1.set_xticks([])
    ax1.set_yticks([])
    ax2.set_xticks([])
    ax2.set_yticks([])
    ax3.set_xticks([])
    ax3.set_yticks([])
    ax1.imshow(image, 'gray')
    ax2.imshow(transf_image, 'gray')
    ax1.title.set_text(title_1)
    ax2.title.set_text(title_2)
    ax3.imshow(diff, 'gray')
    ax3.title.set_text("Difference")
    plt.show()
```

1.1 Q Value plots

[ ]:
```
rng = np.random.default_rng()
idx = rng.choice(np.arange(len(q_values_smdp)), 25, replace=False)
```

[ ]:
```
image1=q_values_smdp[idx]
#image1 = rng.choice(image1, 50)
image2=q_values_intra[idx]
#image2 = rng.choice(image2, 50)AC
diff=np.abs(image1-image2)
plotter(image1,image2,diff,"SMDP","Intra-option")
```



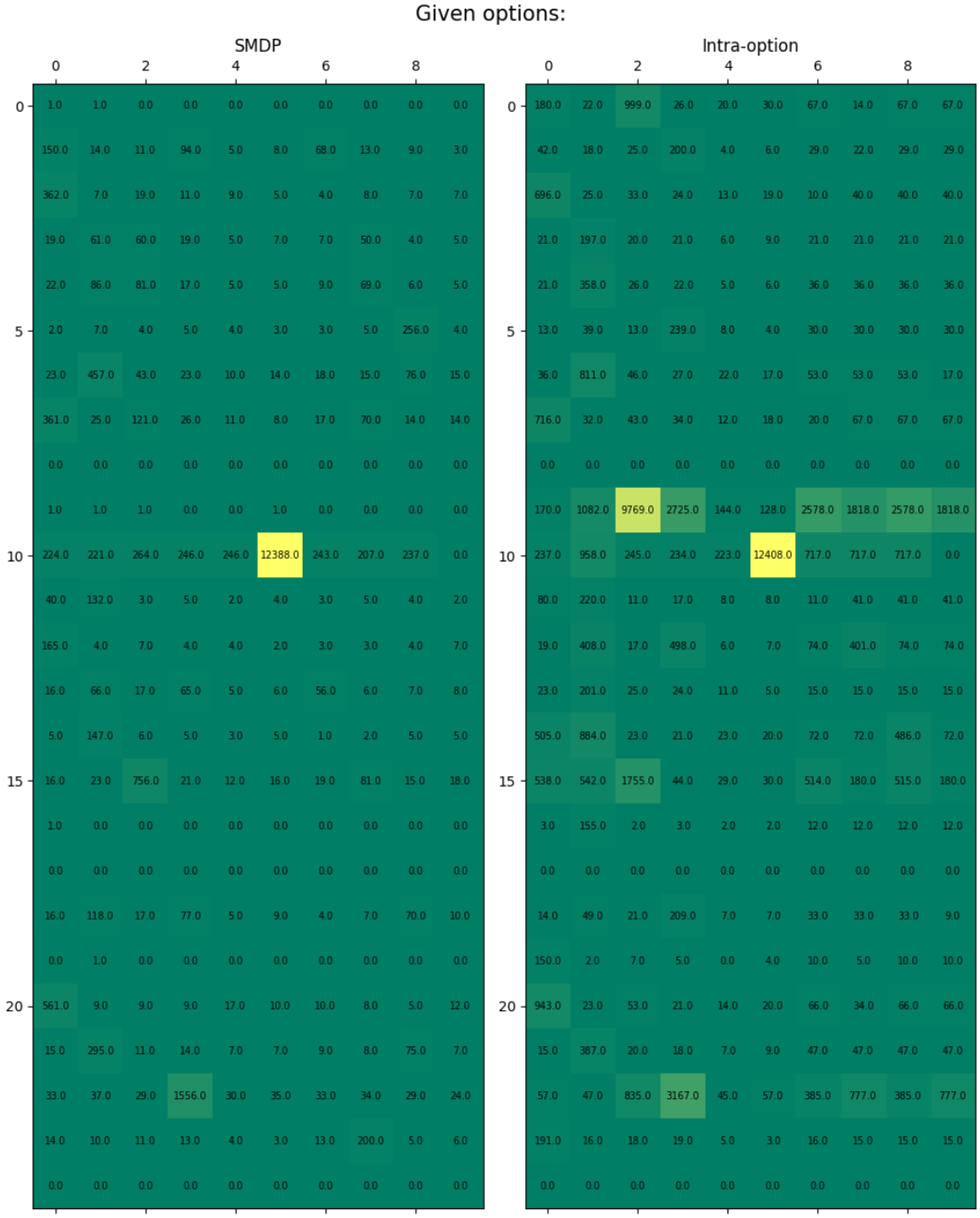
1.1.1 Update frequency:

[ ]:
```
print("Total number of updates in SMDP: "+str(np.sum(uf_values_smdp)))
print("Total number of updates in Intra-option: "+str(np.sum(uf_values_intra)))
```

```
Total number of updates in SMDP: 354452.0
Total number of updates in Intra-option: 1485713.0
```

[ ]:
```
def mat_plotter(image, transf_image):
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,13))
    fig. suptitle("Given options:", fontsize=15)
    ax1.matshow(image,cmap="summer")
    for (i, j), z in np.ndenumerate(image):
      ax1.text(j, i, '{:0.1f}'.format(z), ha='center',␣
 ↪va='center',fontsize="x-small")
    ax2.matshow(transf_image,cmap="summer")
    for (i, j), z in np.ndenumerate(transf_image):
      ax2.text(j, i, '{:0.1f}'.format(z), ha='center',␣
 ↪va='center',fontsize="x-small")
    ax1.title.set_text("SMDP")
    ax2.title.set_text("Intra-option")
    plt.show()
```

[ ]:
```
mat_plotter(uf_values_smdp[idx],uf_values_intra[idx])
```


Given options:
SMDP
Intra-option

[ ]:

CS6700_Assignment3_taxi_part2run

April 22, 2023

0.1 Installation And Imports

[ ]:
```
# installation of older gym for render()
!pip install gym==0.23.1

# some imports and code to ignore deprecation warning
import numpy as np
import random
import gym
import glob
import io
import matplotlib.pyplot as plt
from IPython.display import HTML, display, clear_output
from time import sleep
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting gym==0.23.1
  Downloading gym-0.23.1.tar.gz (626 kB)
                           626.2/626.2 kB
22.8 MB/s eta 0:00:00
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
Requirement already satisfied: importlib-metadata>=4.10.0 in
/usr/local/lib/python3.9/dist-packages (from gym==0.23.1) (6.4.1)
Requirement already satisfied: gym-notices>=0.0.4 in
/usr/local/lib/python3.9/dist-packages (from gym==0.23.1) (0.0.8)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.9/dist-
packages (from gym==0.23.1) (1.22.4)
Requirement already satisfied: cloudpickle>=1.2.0 in
/usr/local/lib/python3.9/dist-packages (from gym==0.23.1) (2.2.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-
packages (from importlib-metadata>=4.10.0->gym==0.23.1) (3.15.0)
Building wheels for collected packages: gym
```

```
  Building wheel for gym (pyproject.toml) … done
  Created wheel for gym: filename=gym-0.23.1-py3-none-any.whl size=701374
sha256=f214e0bf68d73200cc002863767e8e0b8972de315bec0b6a71acd27296a7ac38
  Stored in directory: /root/.cache/pip/wheels/4e/be/7e/92a54668db96883e38ce60a9
249dc55de7cd6eee49e7311940
Successfully built gym
Installing collected packages: gym
  Attempting uninstall: gym
    Found existing installation: gym 0.25.2
    Uninstalling gym-0.25.2:
      Successfully uninstalled gym-0.25.2
Successfully installed gym-0.23.1
```

[ ]:
```
# mount drive
from google.colab import drive
drive.mount('/content/drive')
```

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

0.2 Exploring the environment and rendering a manual episode

[ ]:
```
# explore the taxi-v3 environment to know the observation space(states),␣
 ↪actions, rewards etc
# make env and seed it for reproducibility
env = gym.make('Taxi-v3')
env.seed(42)

# number of states
NUM_STATES = env.observation_space.n
print(f'State count - {NUM_STATES}')

# number of actions
NUM_ACTIONS = env.action_space.n
print(f'Action count - {NUM_ACTIONS}')

ACTION_NAMES = ["South", "North", "East", "West", "Pickup", "Dropoff"]
COLOR_NUM_MAPPING = {0 : "Red", 1 : "Green", 2 : "Yellow", 3 : "Blue"}

state = env.reset()
# state decodes as -> (car row, car col, pass loc, dest); pass loc is a color /␣
 ↪4 : "in car"
print(f'Initial state - {list(env.decode(state))}')
env.render()

# STARTING PASSENGER LOCATION -> BLUE COLOR; DEST -> RED COLOR; Taxi -> GREEN␣
 ↪SQUARE
```

```
# Solving the environment manually to understand it better;
for _ in range(3):
    new_state, reward, done, _ = env.step(1)
    env.render()
    print(f'performed {ACTION_NAMES[1]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
new_state, reward, done, _ = env.step(4)
env.render()
print(f'performed {ACTION_NAMES[4]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
for _ in range(2):
    new_state, reward, done, _ = env.step(0)
    env.render()
    print(f'performed {ACTION_NAMES[0]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
for _ in range(4):
    new_state, reward, done, _ = env.step(3)
    env.render()
    print(f'performed {ACTION_NAMES[3]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
for _ in range(2):
    new_state, reward, done, _ = env.step(0)
    env.render()
    print(f'performed {ACTION_NAMES[0]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
new_state, reward, done, _ = env.step(5)
env.render()
print(f'performed {ACTION_NAMES[5]}; Current state - {list(env.
 ↪decode(new_state))}; reward : {reward}; done : {done}')
```

```
State count - 500
Action count - 6
Initial state - [3, 4, 1, 2]
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | :▒|
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : :▒|
| | : | : |
|Y| : |B: |
```

```
+---------+
  (North)
performed North; Current state - [2, 4, 1, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : :_|
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
performed North; Current state - [1, 4, 1, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
performed North; Current state - [0, 4, 1, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
performed Pickup; Current state - [0, 4, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : :_|
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
performed South; Current state - [1, 4, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : :_|
| | : | : |
|Y| : |B: |
+---------+
  (South)
performed South; Current state - [2, 4, 4, 2]; reward : -1; done : False
```

```
+---------+
|R: | : :G|
| : | : : |
| : : :_: |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 3, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : :_: : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 2, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| :_: : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 1, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
performed West; Current state - [2, 0, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (South)
performed South; Current state - [3, 0, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
```

```
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
performed South; Current state - [4, 0, 4, 2]; reward : -1; done : False
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
performed Dropoff; Current state - [4, 0, 2, 2]; reward : 20; done : True
```

0.3 Plotting and Helper Functions

[ ]:
```
class bcolors:
    RED= '\u001b[31m'
    GREEN= '\u001b[32m'
    RESET= '\u001b[0m'

def see_trained_agent(qtable, stime=1.5):
    episodes_to_preview = 3
    env = gym.make('Taxi-v3')
    env.seed(10)
    opt_func_map = {6: Drive_to_PX, 7 : Drive_to_PY} # opt id to option␣
 ↪function map
    for episode in range(episodes_to_preview):

        # Reset the environment
        state = env.reset()
        clear_output(wait=True); print(f"TRAINED AGENT\n+++++EPISODE␣
 ↪{episode+1}++++++"); env.render()
        eps_reward = 0
        if eps_reward < 0:
            print(f"Score: {bcolors.RED}{eps_reward}{bcolors.RESET}")
        else:
            print(f"Score: {bcolors.GREEN}{eps_reward}{bcolors.RESET}")
        sleep(stime)
        done = False

        while not done:
            # Exploit
            action = np.argmax(qtable[state])
```

```
            # Take an action and observe the reward
            if action < 6:
                next_state, reward, done, info = env.step(action)
                eps_reward += reward
                state = next_state
                clear_output(wait=True); print(f"TRAINED AGENT\n+++++EPISODE␣
 ↪{episode+1}++++++")
                env.render()
                if eps_reward < 0:
                    print(f"Score: {bcolors.RED}{eps_reward}{bcolors.RESET}")
                else:
                    print(f"Score: {bcolors.GREEN}{eps_reward}{bcolors.RESET}")
                sleep(stime)
            else:
                optdone = False
                opt_func = opt_func_map[action]
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    eps_reward += reward
                    _, optdone = opt_func(env, next_state)
                    state = next_state
                    clear_output(wait=True); print(f"TRAINED␣
 ↪AGENT\n+++++EPISODE {episode+1}++++++")
                    env.render()
                    if eps_reward < 0:
                        print(f"Score: {bcolors.RED}{eps_reward}{bcolors.
 ↪RESET}")
                    else:
                        print(f"Score: {bcolors.GREEN}{eps_reward}{bcolors.
 ↪RESET}")
                    sleep(stime)

# function to test an agent after training
def test_agent(q_values, test_episodes):
    env = gym.make('Taxi-v3')
    env.seed(0)
    test_rewards = []
    opt_func_map = {6: Drive_to_PX, 7 : Drive_to_PY} # opt id to option␣
 ↪function map
    for _ in tqdm(range(test_episodes)):
        # Reset the environment
        state = env.reset()
        eps_reward = 0
        done = False
        while not done:
            # Exploit
```

```
            action = np.argmax(q_values[state])

            # Take an action and observe the reward
            if action < 6:
                next_state, reward, done, info = env.step(action)
                eps_reward += reward
                state = next_state
            else:
                optdone = False
                opt_func = opt_func_map[action]
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    eps_reward += reward
                    _, optdone = opt_func(env, next_state)
                    state = next_state
        test_rewards.append(eps_reward)
    return test_rewards
```

[ ]:
```
MAINPATH = '/content/drive/MyDrive/Colab Notebooks/CS6700/Assignments/
 ↪Assignment3/Part2'
def plot_reward_curve(eps_reward_list, path, alp, eps, gam=0.9):
    fig, ax = plt.subplots(figsize=(16,8))
    ax.set_title('Reward Curve')
    ax.set_xlabel('Episode Number')
    ax.set_ylabel('Reward')
    num_episodes = len(eps_reward_list)
    ax.plot(np.arange(1, num_episodes+1), eps_reward_list, color='red')
    props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
    textstr = ' alpha = {}\n epsilon = {}\n gamma = {}'.format(alp,eps,gam)
    ax.text(0.8, 0.2, textstr, transform=ax.transAxes, fontsize=12,␣
 ↪verticalalignment='top', bbox=props)
    fig.savefig(MAINPATH+'/'+path+'/'+'reward_curve.png')
```

[ ]:
```
def visualize_q_values(q_values, msg, path, pass_src = None, pass_dest = None):
    assert(pass_src != None and pass_dest != None)

    req_actions = [[None for _ in range(5)] for _ in range(5)]
    req_q_values = [[None for _ in range(5)] for _ in range(5)]
    temp_env = gym.make('Taxi-v3')
    for s in range(500):
        s_vec = list(temp_env.decode(s))
        if s_vec[2] == pass_src and s_vec[3] == pass_dest:
            req_actions[s_vec[0]][s_vec[1]] = np.argmax(q_values[s])
            req_q_values[s_vec[0]][s_vec[1]] = np.max(q_values[s])
    req_actions = np.array(req_actions)
    fig, ax = plt.subplots(figsize=(5,5))
```

```
    ax.set_title(msg)
    ax.invert_yaxis()
    ax.xaxis.tick_top()
    ax.set_xlabel('Columns')
    ax.xaxis.set_label_position('top')
    ax.set_ylabel('Rows')
    mesh = ax.pcolormesh(req_q_values, edgecolors='k', linewidths=2)
    fig.colorbar(mesh)
    def x_direct(a):
        if a in [4,5,6,7,8]:
            return 0
        if a in [0, 1]:
            return 0
        return 1 if a == 2 else -1
    def y_direct(a):
        if a in [4,5,6,7,8]:
            return 0
        if a in [2, 3]:
            return 0
        return 1 if a == 1 else -1
    idx = np.indices((5,5))
    policyx = np.vectorize(x_direct)(req_actions)
    policyy = np.vectorize(y_direct)(req_actions)
    req_action_dict = {4 : 'Pickup', 5 : 'Drop', 6 : 'Goto_PX', 7 : 'Goto_PY'}
    for i,j,px,py in zip(idx[1].ravel(), idx[0].ravel(), policyx.ravel(),␣
↪policyy.ravel()):
        if (req_actions[j, i] < 4):
            ax.quiver(i+0.5, j+0.5, px, py, pivot="middle",␣
↪scale=10,color='red')
        else:
            ax.text(i+0.5, j+0.5, req_action_dict[req_actions[j][i]],␣
↪horizontalalignment='center',verticalalignment='center',color='tomato')
    fig.savefig(MAINPATH+'/'+path+'/'+msg+'.png')
```

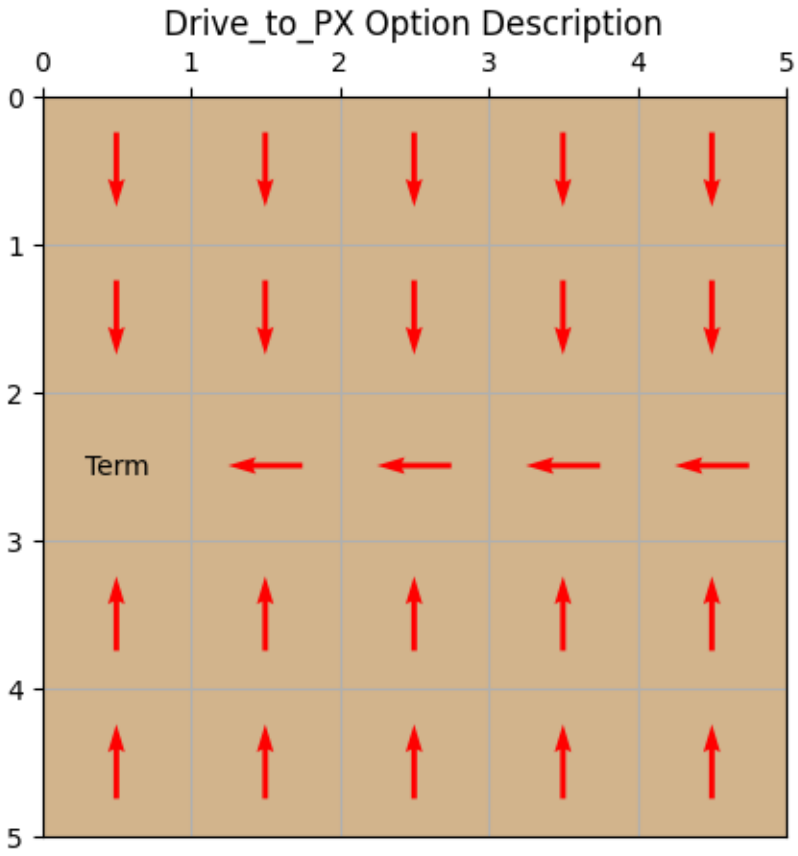
0.4 Defining the Reach Main Road, Drive Left on Main Road, Drive Right on Main Road Options

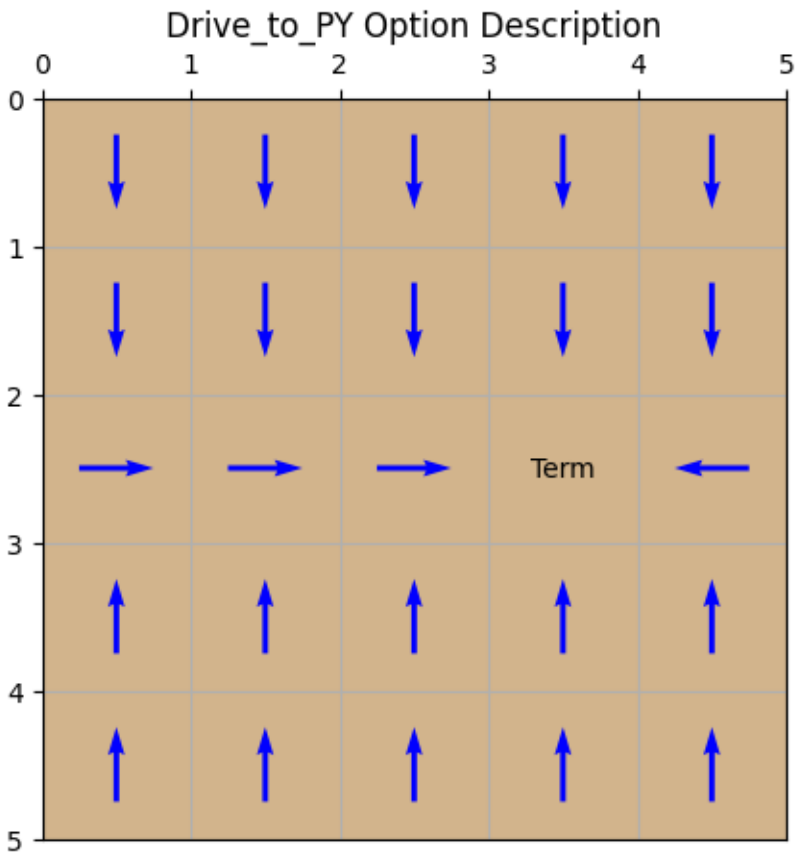
[ ]:
```
PLOTSPATH = MAINPATH + '/OptionDescs'
def save_option_desc(policy, desc, color, end):
    pcopy = np.copy(policy)
    pcopy[end] = -1
    fig, ax = plt.subplots(figsize = (5,5))
    ax.set_title(f'{desc} Option Description')
    ax.set_xlim([0,5])
    ax.set_ylim([0,5])
    def x_direct(a):
```

```
        if a in [-1, 0, 1]:
            return 0
        return 1 if a == 2 else -1
    def y_direct(a):
        if a in [-1, 2, 3]:
            return 0
        return 1 if a == 1 else -1
    policyx = np.vectorize(x_direct)(pcopy)
    policyy = np.vectorize(y_direct)(pcopy)
    idx = np.indices(policy.shape)
    for i,j,px,py in zip(idx[1].ravel(), idx[0].ravel(), policyx.ravel(),␣
 ↪policyy.ravel()):
        if (pcopy[j, i] != -1):
            ax.quiver(i+0.5, j+0.5, px, py, pivot="middle",␣
 ↪scale=10,color=color)
        else:
            ax.text(i+0.5, j+0.5, 'Term',␣
 ↪horizontalalignment='center',verticalalignment='center')
    ax.set_facecolor('tan'); ax.invert_yaxis(); ax.xaxis.tick_top(); ax.grid()
    fig.savefig(f'{PLOTSPATH}/option_{desc}.png')
```

[ ]:
```
# defining the required option action matrix for all the options [FOR EASY␣
 ↪CODING]
X_ACTN_MATRIX = np.array([[0,0,0,0,0],
                          [0,0,0,0,0],
                          [0,3,3,3,3],
                          [1,1,1,1,1],
                          [1,1,1,1,1]])
X_TERM_MATRIX = np.array([[0,0,0,0,0],
                          [0,0,0,0,0],
                          [1,0,0,0,0],
                          [0,0,0,0,0],
                          [0,0,0,0,0]])
save_option_desc(X_ACTN_MATRIX,'Drive_to_PX', 'red', (2,0))
Y_ACTN_MATRIX = np.array([[0,0,0,0,0],
                          [0,0,0,0,0],
                          [2,2,2,2,3],
                          [1,1,1,1,1],
                          [1,1,1,1,1]])
Y_TERM_MATRIX = np.array([[0,0,0,0,0],
                          [0,0,0,0,0],
                          [0,0,0,1,0],
                          [0,0,0,0,0],
                          [0,0,0,0,0]])
save_option_desc(Y_ACTN_MATRIX, 'Drive_to_PY', 'blue', (2,3))

OPT_TO_POLICY_MAP = {6 : X_ACTN_MATRIX, 7 : Y_ACTN_MATRIX}
```

```
OPT_TO_TERM_MAP = {6 : X_TERM_MATRIX, 7 : Y_TERM_MATRIX}
```





[ ]:
```
# defining the 2 Option functions
def Drive_to_PX(env, state):
    coords = list(env.decode(state))[:2]
    optdone = False
    optact = X_ACTN_MATRIX[coords[0], coords[1]]

    if (coords == [2, 0]): # termination at reaching cell X
        optdone = True

    return [optact, optdone]

def Drive_to_PY(env, state):
    coords = list(env.decode(state))[:2]
    optdone = False
    optact = Y_ACTN_MATRIX[coords[0], coords[1]]

    if (coords == [2, 3]): # termination at reaching cell Y
        optdone = True
```

```
    return [optact, optdone]
```

0.5 Exploration Function

[ ]:
```
#Q-Table: (States x Actions) === (env.ns(500) x total actions(8))
q_values_test = np.zeros((500, 8))
uf_values_test = np.zeros((500, 8)) # Update_Frequency Data structure - stores␣
↪the number of updates for (s,a) where a = prim action/an option
avl_actions = [0, 1, 2, 3, 4, 5, 6, 7] # for ["South", "North", "East", "West",␣
↪"Pickup", "Drop", "Drive_to_X" opt, "Drive_to_Y" opt]

def egreedy_policy(q_values, state, epsilon, rg, disallow):
    if (rg.random() < epsilon):   # epsilon prob for uniform choice over all␣
↪actions and options
        if (disallow != None):
            val_actions = avl_actions[:]; val_actions.remove(disallow)
            return rg.choice(val_actions)
        else:
            return rg.choice(avl_actions)
    else:                         # 1 - epsilon prob for greedy action/option
            return np.argmax(q_values[state])
```

0.6 SMDP Q-Learning

[ ]:
```
#### SMDP Q-Learning

def SMDP_QLearning(alpha, epsilon, epcount = 10000, gamma = 0.9, rg = np.random.
↪RandomState(42)):
    env = gym.make('Taxi-v3')
    env.seed(42) # for reproducibility

    # arrays for storing q value, update counts and episode rewards
    q_values = np.zeros((500, 8))
    uf_values = np.zeros((500, 8))
    ep_rewards_list = []
    opt_func_map = {6: Drive_to_PX, 7 : Drive_to_PY} # opt id to option␣
↪function map
    dis_opts = {(2,0) : 6, (2,3) : 7} # store disallowed options for each color␣
↪coordinates in a dict

    # Iterate over epcount episodes
    for epidx in tqdm(range(epcount)):
        state = env.reset()
        done = False
        eps_reward = 0
```

```
        # While episode is not over
        while not done:
            # Choose action/option. Note : option should be allowed at the␣
↪current co-ordinates
            st_coords = tuple(env.decode(state))[:2]
            dis_opt = (dis_opts[st_coords] if st_coords in dis_opts.keys() else␣
↪None)
            action = egreedy_policy(q_values, state, epsilon, rg, dis_opt)
            # Checking if primitive action
            if action < 6:
                # Perform regular Q-Learning update for state-action pair
                next_state, reward, done, _ = env.step(action)
                q_values[state, action] += alpha * (reward + gamma*np.
↪max(q_values[next_state]) - q_values[state, action])
                uf_values[state,action] += 1
                state = next_state
                eps_reward += reward
            else:
                # find the corresponding function for the option
                opt_func = opt_func_map[action]
                reward_bar = 0

                initial_state = state
                opt_steps = 0
                optdone = False
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    _, optdone = opt_func(env, next_state)
                    reward_bar = gamma * reward_bar + reward
                    opt_steps += 1
                    state = next_state
                    eps_reward += reward

                # SMDP Q-Learning Update for options. We update the q-value of␣
↪the (initial_state, option) pair at the end of option execution.
                q_values[initial_state, action] += alpha * (reward_bar + (gamma␣
↪** opt_steps) * np.max(q_values[state]) - q_values[initial_state, action])
                uf_values[initial_state,action] += 1
        ep_rewards_list.append(eps_reward)
    return q_values, uf_values, ep_rewards_list
```

0.7 Intra Option Q-Learning

[ ]:
```
#### Intra Option Q-Learning

# NOT USED - INSIGNIFICANT SPEED BOOST
def gen_policies_sharing_action():
    ans = [[[[] for _ in range(6)] for _ in range(5)] for _ in range(5)]
    for opt in [6, 7]:
        policy = OPT_TO_POLICY_MAP[opt]
        for i in range(5):
            for j in range(5):
                ans[i][j][policy[i,j]].append(opt)
    return ans

def IntraOption_QLearning(alpha, epsilon, epcount = 10000, gamma = 0.9, rg = np.
 ↪random.RandomState(42)):
    env = gym.make('Taxi-v3')
    env.seed(42) # for reproducibility

    # arrays for storing q value, update counts and episode rewards
    q_values = np.zeros((500, 8))
    uf_values = np.zeros((500, 8))
    ep_rewards_list = []
    opt_func_map = {6: Drive_to_PX, 7 : Drive_to_PY} # opt id to option␣
 ↪function map
    dis_opts = {(2,0) : 6, (2,3) : 7} # store disallowed options for each color␣
 ↪coordinates in a dict

    # Iterate over epcount episodes
    for epidx in tqdm(range(epcount)):
        state = env.reset()
        done = False
        eps_reward = 0

        # While episode is not over
        while not done:
            # Choose action/option. Note : option should be allowed at the␣
 ↪current co-ordinates
            st_coords = tuple(env.decode(state))[:2]
            dis_opt = (dis_opts[st_coords] if st_coords in dis_opts.keys() else␣
 ↪None)
            action = egreedy_policy(q_values, state, epsilon, rg, dis_opt)
            # Checking if primitive action
            if action < 6:
                # Perform regular Q-Learning update for state-action pair
                next_state, reward, done, _ = env.step(action)
```

```
                q_values[state, action] += alpha * (reward + gamma*np.
↪max(q_values[next_state]) - q_values[state, action])
                uf_values[state,action] += 1
                state = next_state
                eps_reward += reward
            else:
                # find the corresponding function for the option
                opt_func = opt_func_map[action]
                other_opts = [6,7]; other_opts.remove(action)
                optdone = False
                while (optdone == False):
                    optact, _ = opt_func(env, state)
                    next_state, reward, done, _ = env.step(optact)
                    _, optdone = opt_func(env, next_state)
                    eps_reward += reward
                    # perform update for the optact - primitive action
                    q_values[state, optact] += alpha * (reward + gamma*np.
↪max(q_values[next_state]) - q_values[state, optact])
                    uf_values[state, optact] += 1

                    # find all options that choose the same primitive action at␣
↪state
                    opt_upd_list = [action]
                    st_coords = list(env.decode(state))[:2]
                    for oth in other_opts:
                        if OPT_TO_POLICY_MAP[oth][st_coords[0], st_coords[1]]␣
↪== optact:
                            opt_upd_list.append(oth)
                    # debug - print(st_coords, action, opt_upd_list)
                    # perform updates for all options that choose the same␣
↪primitive action at state
                    nst_coords = list(env.decode(next_state))[:2]
                    for opt in opt_upd_list:
                        term_matrix = OPT_TO_TERM_MAP[opt]
                        if term_matrix[nst_coords[0], nst_coords[1]] == 1:
                            # if the option terminates, we do total max over␣
↪all actions(and options) in next state
                            q_values[state, opt] += alpha * (reward + gamma *␣
↪np.max(q_values[next_state]) - q_values[state, opt])
                            uf_values[state, opt] += 1
                        else:
                            # if it does not, we use the option q-value in next␣
↪state for update
                            q_values[state, opt] += alpha * (reward + gamma *␣
↪(q_values[next_state, opt]) - q_values[state, opt])
                            uf_values[state, opt] += 1
```

```
                state = next_state
            ep_rewards_list.append(eps_reward)
        return q_values, uf_values, ep_rewards_list
```

0.8 Wandb Sweeping For finding best hyperparameters

[ ]:
```
!pip install wandb
import wandb
wandb.login()
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting wandb
  Downloading wandb-0.14.2-py3-none-any.whl (2.0 MB)
                             2.0/2.0 MB
31.3 MB/s eta 0:00:00
Requirement already satisfied: psutil>=5.0.0 in
/usr/local/lib/python3.9/dist-packages (from wandb) (5.9.4)
Collecting docker-pycreds>=0.4.0
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
Requirement already satisfied: Click!=8.0.0,>=7.0 in
/usr/local/lib/python3.9/dist-packages (from wandb) (8.1.3)
Collecting pathtools
  Downloading pathtools-0.1.2.tar.gz (11 kB)
  Preparing metadata (setup.py) … done
Collecting setproctitle
  Downloading setproctitle-1.3.2-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_6
4.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (30 kB)
Collecting sentry-sdk>=1.0.0
  Downloading sentry_sdk-1.19.1-py2.py3-none-any.whl (199 kB)
                            199.2/199.2 kB
25.2 MB/s eta 0:00:00
Requirement already satisfied: requests<3,>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from wandb) (2.27.1)
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.15.0 in
/usr/local/lib/python3.9/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.9/dist-packages (from wandb) (4.5.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-
packages (from wandb) (67.6.1)
Requirement already satisfied: appdirs>=1.4.3 in /usr/local/lib/python3.9/dist-
packages (from wandb) (1.4.4)
Collecting GitPython!=3.1.29,>=1.0.0
  Downloading GitPython-3.1.31-py3-none-any.whl (184 kB)
                            184.3/184.3 kB
23.8 MB/s eta 0:00:00
```

```
Requirement already satisfied: PyYAML in /usr/local/lib/python3.9/dist-
packages (from wandb) (6.0)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.9/dist-
packages (from docker-pycreds>=0.4.0->wandb) (1.16.0)
Collecting gitdb<5,>=4.0.1
  Downloading gitdb-4.0.10-py3-none-any.whl (62 kB)
                             62.7/62.7 kB
7.5 MB/s eta 0:00:00
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.0.0->wandb) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.0.0->wandb)
(1.26.15)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.0.0->wandb) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.0.0->wandb)
(2022.12.7)
Collecting smmap<6,>=3.0.1
  Downloading smmap-5.0.0-py3-none-any.whl (24 kB)
Building wheels for collected packages: pathtools
  Building wheel for pathtools (setup.py) … done
  Created wheel for pathtools: filename=pathtools-0.1.2-py3-none-any.whl
size=8807
sha256=37bfe1832b91063a132edffc63db5238f1a7a59f9e4144509f50fff16fd18bbe
  Stored in directory: /root/.cache/pip/wheels/b7/0a/67/ada2a22079218c75a88361c0
782855cc72aebc4d18d0289d05
Successfully built pathtools
Installing collected packages: pathtools, smmap, setproctitle, sentry-sdk,
docker-pycreds, gitdb, GitPython, wandb
Successfully installed GitPython-3.1.31 docker-pycreds-0.4.0 gitdb-4.0.10
pathtools-0.1.2 sentry-sdk-1.19.1 setproctitle-1.3.2 smmap-5.0.0 wandb-0.14.2
```

```
<IPython.core.display.Javascript object>
```

```
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
```

[ ]:
```
True
```

0.8.1 SMDP Q-Learning Hyperparameter Finetuning

[ ]:
```python
def test_smdp():
    run = wandb.init(reinit=True)
    al = wandb.config.alpha
    epsn = wandb.config.epsilon
    epcount = wandb.config.train_episodes
    run.name = 'al={:.4f}_eps={:.4f}'.format(al, epsn)
```

```
    q_values, uf_values, ep_reward_list = SMDP_QLearning(al, epsn, epcount)

    for i, rew in enumerate(ep_reward_list):
        wandb.log({'train episode number': i+1, 'train_reward' : rew})

    test_rewards = test_agent(q_values, wandb.config.test_episodes)
    for i, rew in enumerate(test_rewards):
        wandb.log({'test episode number': i+1, 'test_reward' : rew})

    wandb.log({'avg_test_reward' : np.mean(test_rewards)})
    run.finish()
```

[ ]:
```
sweep_config = {
    'method' : 'bayes',
    'name' : 'alpha_epsilon_sweep',
    'metric' : {
        'goal' : 'maximize',
        'name' : 'avg_test_reward'
    },
    'parameters' : {
        'alpha' : {'min' : 0.2, 'max' : 0.7},
        'epsilon' : {'min' : 0.02, 'max' : 0.25},
        'train_episodes' : {'value' : 20000},
        'test_episodes' : {'value' : 1000}
    }
}

sweep_id = wandb.sweep(sweep=sweep_config, project='RL-A3-P2-SMDP',␣
 ↪entity='cs6700_team_2023')
wandb.agent(sweep_id, test_smdp, 'cs6700_team_2023', count=100)
```

0.8.2 Intra-Option Q-Learning Hyperparameter Finetuning

[ ]:
```
def test_intra():
    run = wandb.init(reinit=True)
    al = wandb.config.alpha
    epsn = wandb.config.epsilon
    epcount = wandb.config.train_episodes
    run.name = 'al={:.4f}_eps={:.4f}'.format(al, epsn)
    q_values, uf_values, ep_reward_list = IntraOption_QLearning(al, epsn,␣
 ↪epcount)

    for i, rew in enumerate(ep_reward_list):
        wandb.log({'train episode number': i+1, 'train_reward' : rew})

    test_rewards = test_agent(q_values, wandb.config.test_episodes)
```

```
    for i, rew in enumerate(test_rewards):
        wandb.log({'test episode number': i+1, 'test_reward' : rew})

    wandb.log({'avg_test_reward' : np.mean(test_rewards)})
    run.finish()
```

[ ]:
```
sweep_config = {
    'method' : 'bayes',
    'name' : 'alpha_epsilon_sweep',
    'metric' : {
        'goal' : 'maximize',
        'name' : 'avg_test_reward'
    },
    'parameters' : {
        'alpha' : {'min' : 0.2, 'max' : 0.7},
        'epsilon' : {'min' : 0.02, 'max' : 0.25},
        'train_episodes' : {'value' : 20000},
        'test_episodes' : {'value' : 1000}
    }
}

sweep_id = wandb.sweep(sweep=sweep_config,␣
 ↪project='RL-A3-P2-IntraOptionQLearning', entity='cs6700_team_2023')
wandb.agent(sweep_id, test_intra, 'cs6700_team_2023', count=100)
```

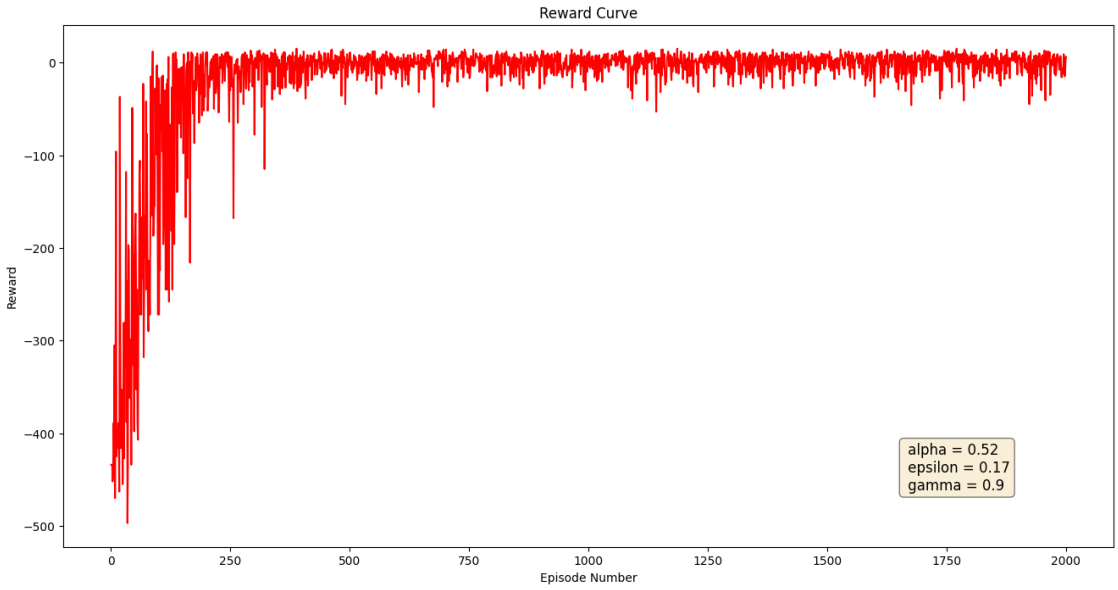
0.9 Generating Plots for Best Hyperparameter Combination

0.9.1 SMDP Q-Learning

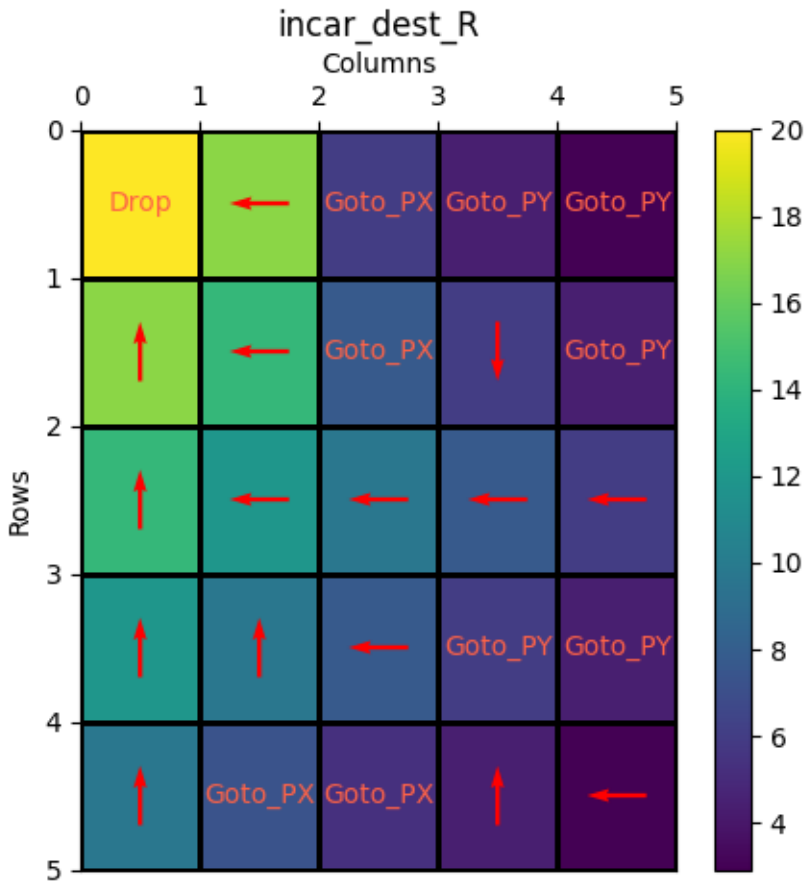
[ ]:
```
bestalpha = 0.52
bestepsilon = 0.17

q_values_smdp, uf_values_smdp, eps_rewards = SMDP_QLearning(bestalpha,␣
 ↪bestepsilon, 50000) # 30k to 50k
```

```
100%|      | 50000/50000 [01:05<00:00, 764.36it/s]
```

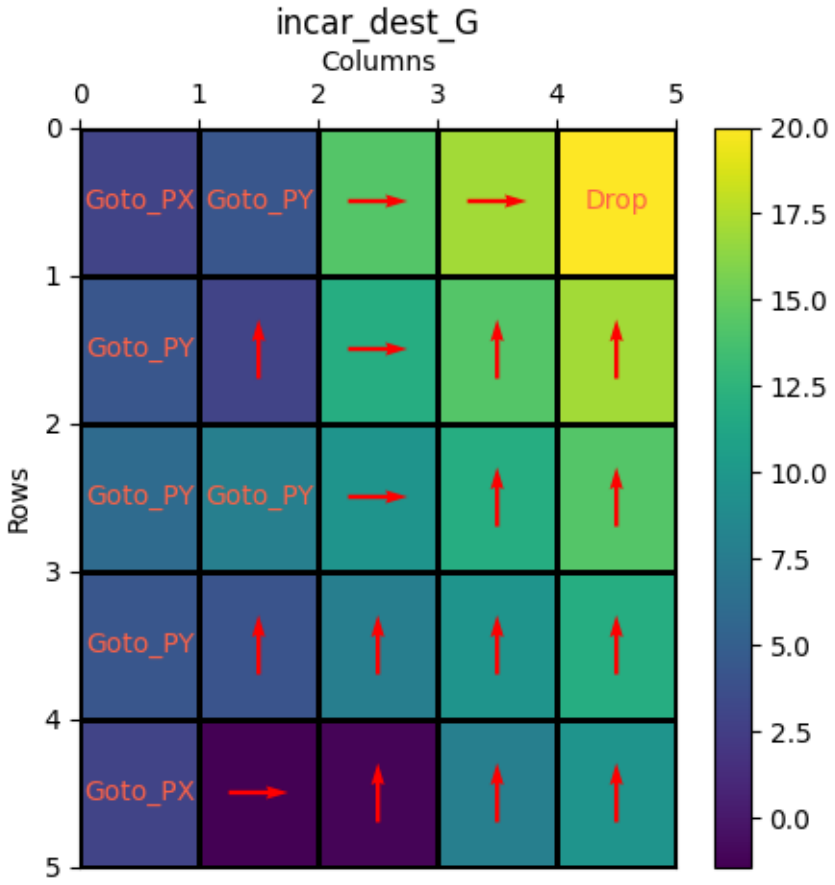
[ ]:
```
plot_reward_curve(eps_rewards[:2000], 'SMDP', bestalpha, bestepsilon) # 2k to 3k
```



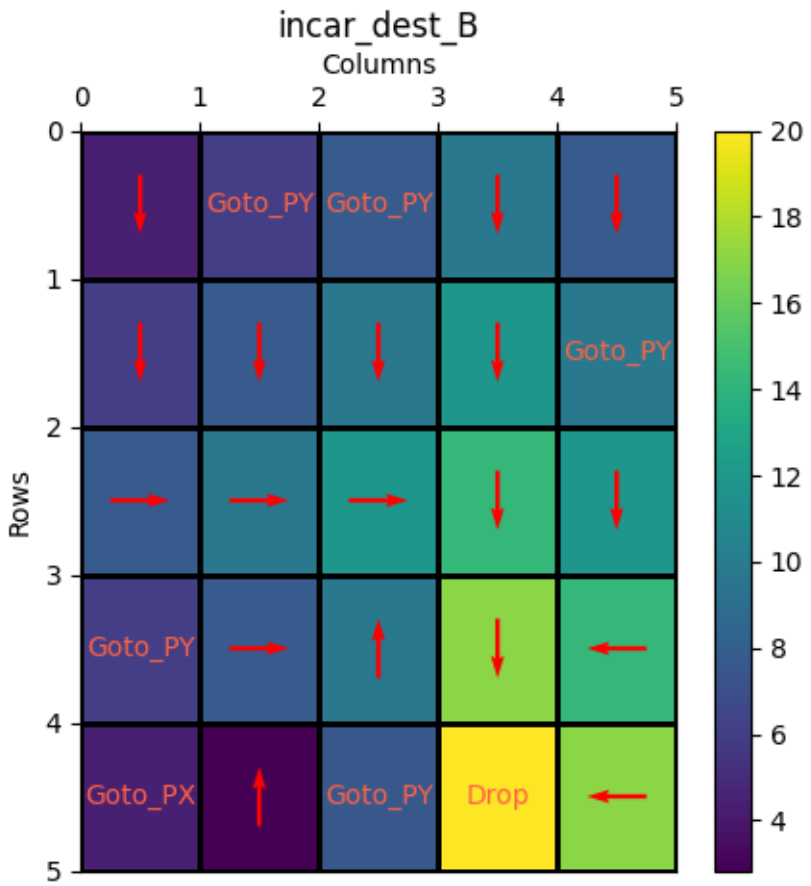
[ ]:
```
visualize_q_values(q_values_smdp, "incar_dest_R", 'SMDP', 4, 0)
```



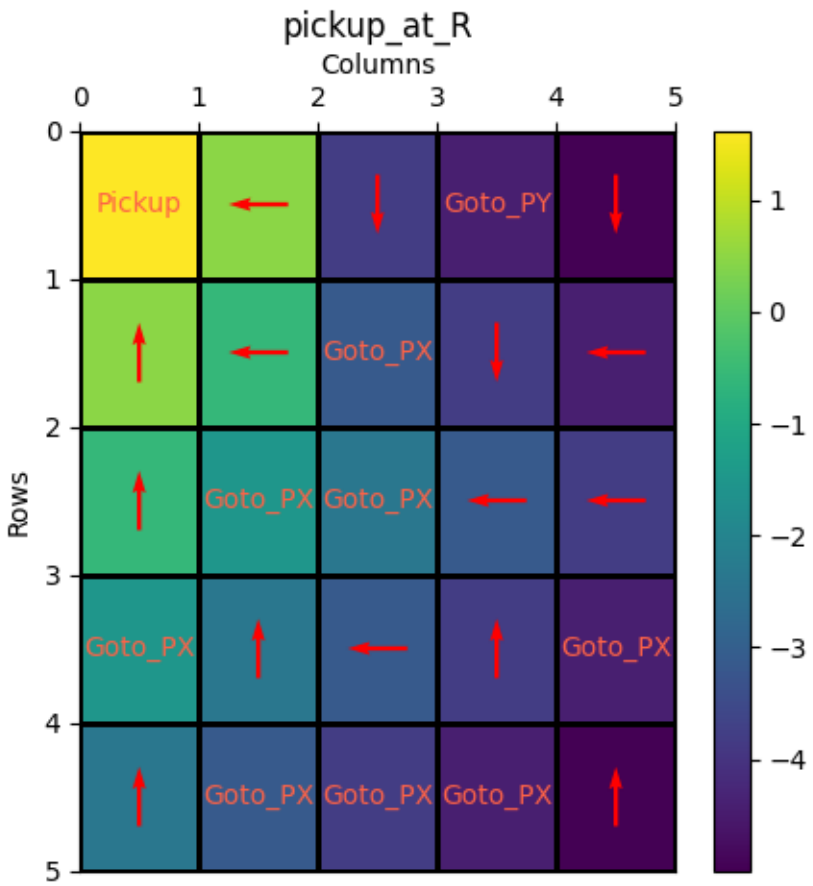
[ ]:
```
visualize_q_values(q_values_smdp, "incar_dest_G", 'SMDP', 4, 1)
```



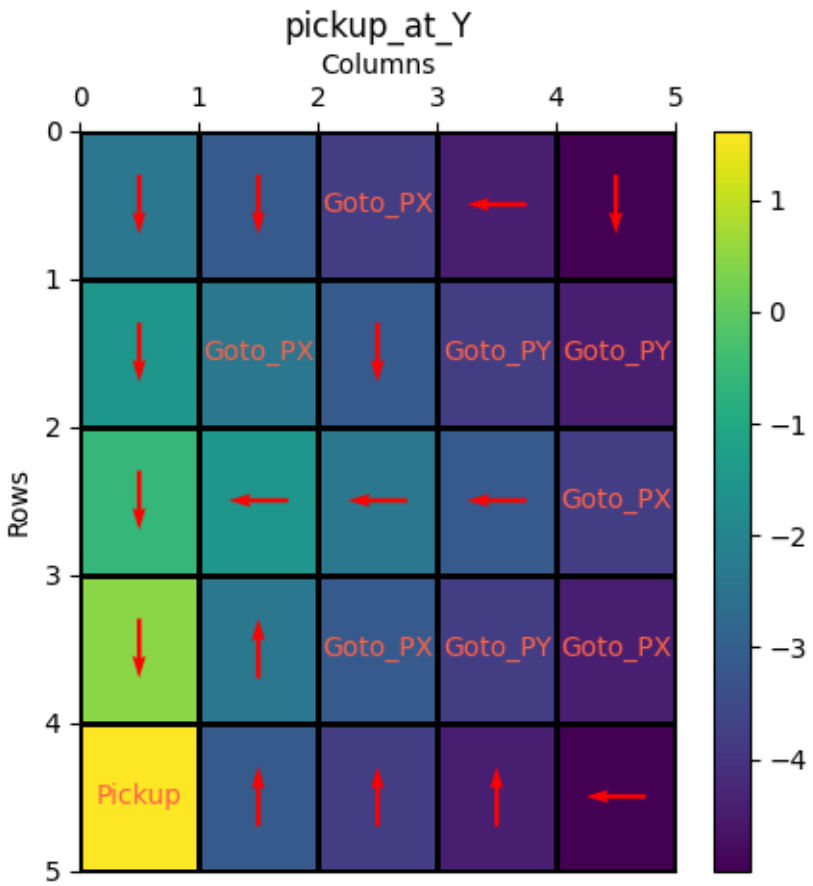
[ ]:
```
visualize_q_values(q_values_smdp, "incar_dest_B", 'SMDP', 4, 3)
```



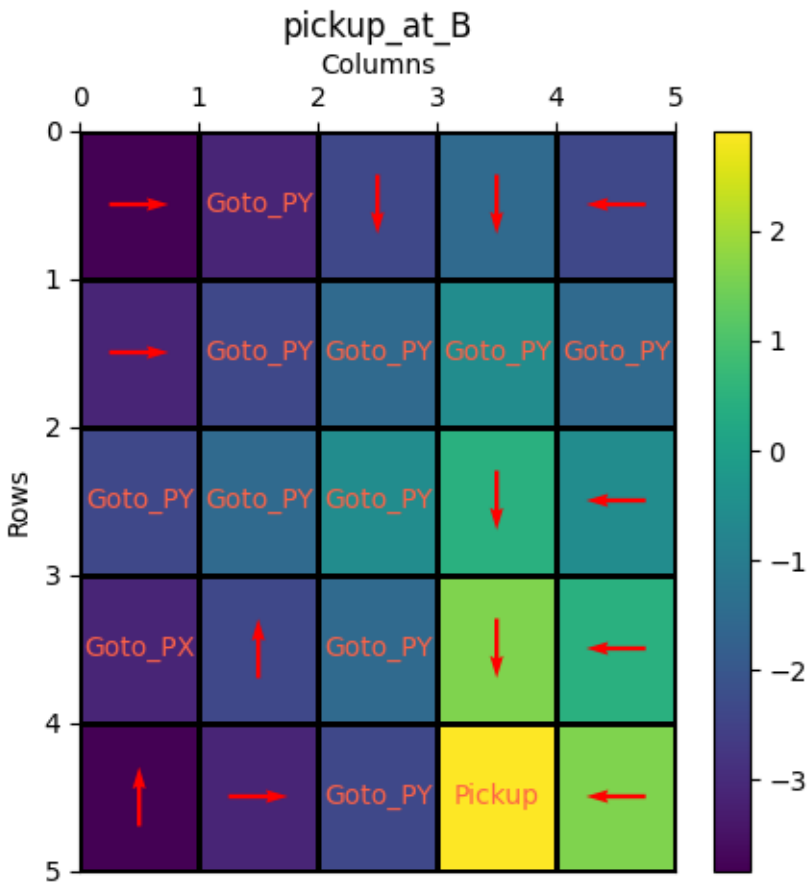
[ ]:
```
visualize_q_values(q_values_smdp, "pickup_at_R", 'SMDP', 0, 1)
```



[ ]:
```
visualize_q_values(q_values_smdp, "pickup_at_Y", 'SMDP', 2, 1)
```



[ ]:
```
visualize_q_values(q_values_smdp, "pickup_at_B", 'SMDP', 3, 2)
```

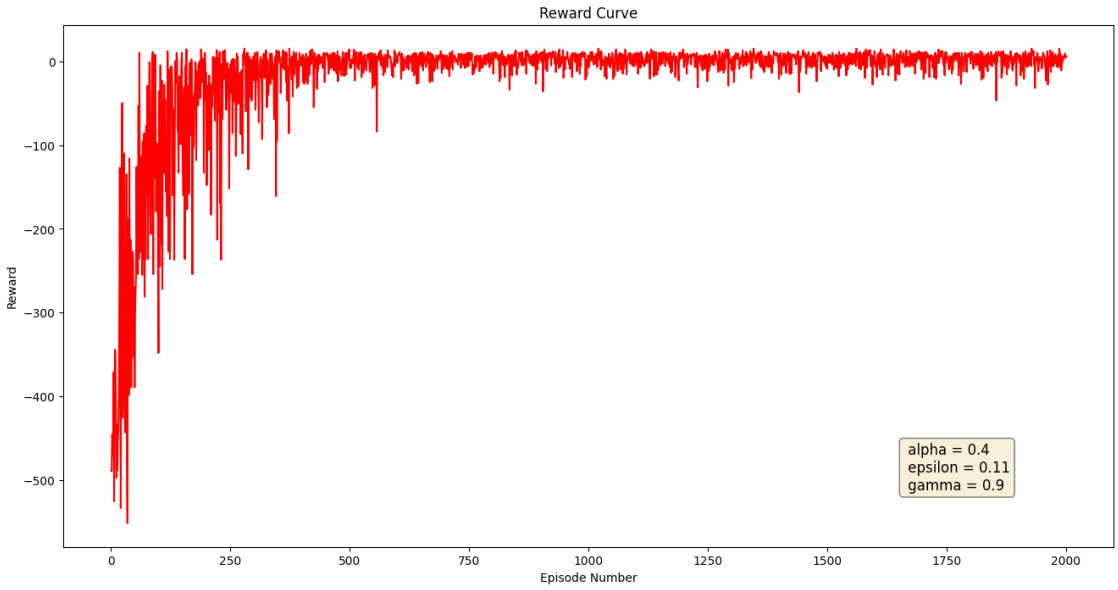


0.9.2 Intra-Option Q-Learning

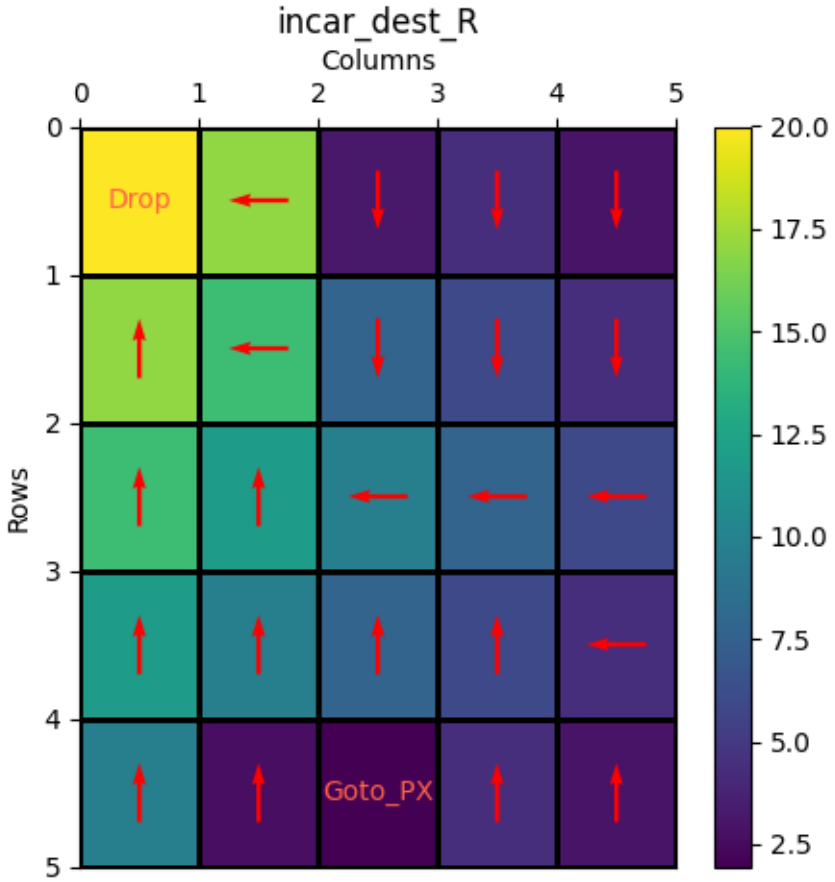
[ ]:
```
bestalpha = 0.4
bestepsilon = 0.11

q_values_intra, uf_values_intra, eps_rewards_intra =␣
↪IntraOption_QLearning(bestalpha, bestepsilon, 50000) # 30k to 50k
```

```
100%|      | 50000/50000 [00:58<00:00, 861.59it/s]
```

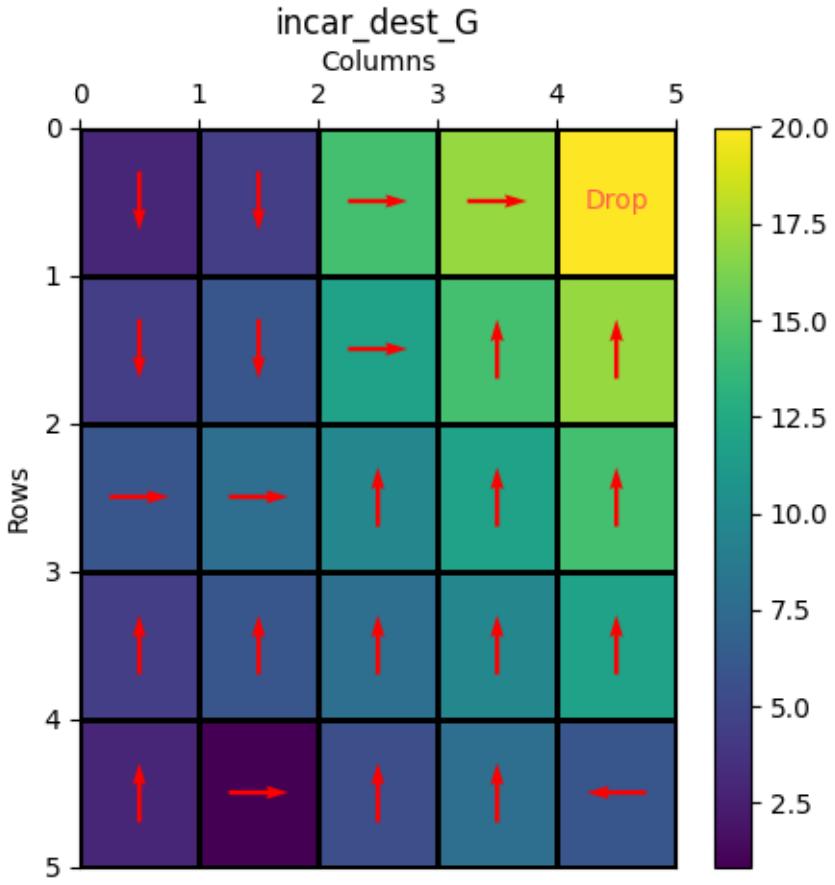
[ ]:
```
plot_reward_curve(eps_rewards_intra[:2000], 'Intra', bestalpha, bestepsilon) #␣
↪2k to 3k
```



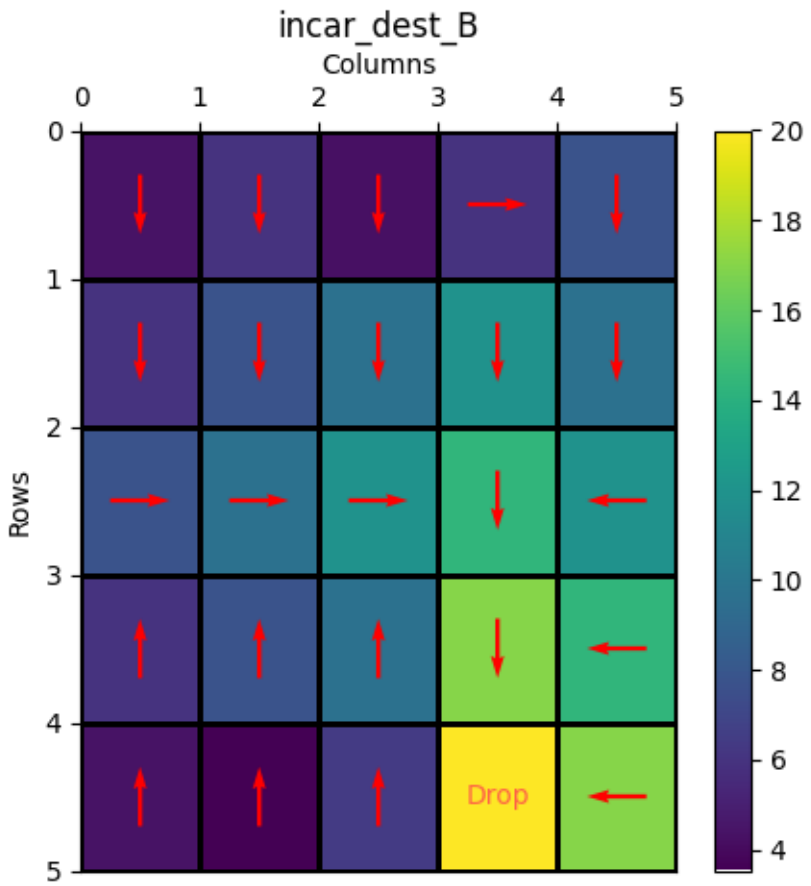
[ ]:
```
visualize_q_values(q_values_intra, "incar_dest_R", 'Intra', 4, 0)
```



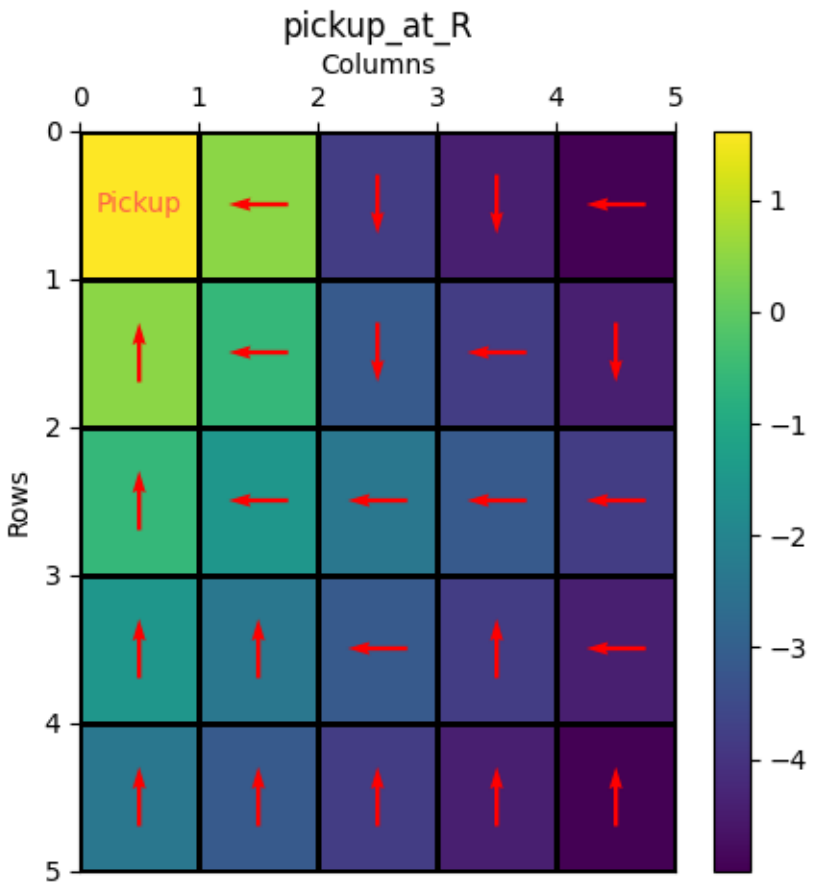
[ ]:
```
visualize_q_values(q_values_intra, "incar_dest_G", 'Intra', 4, 1)
```



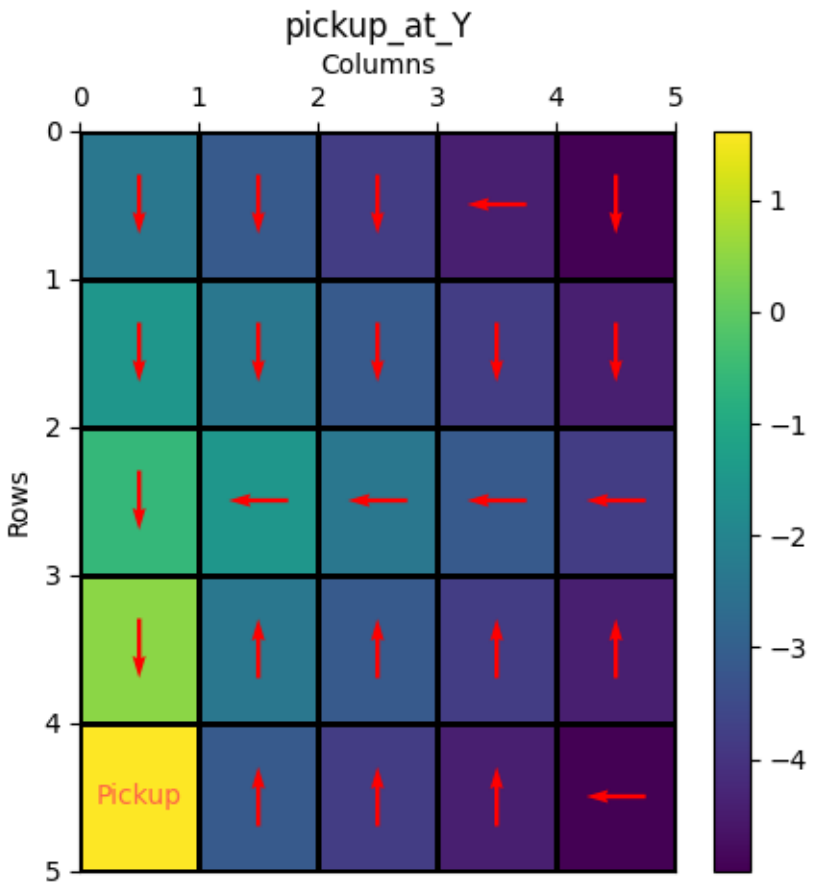
[ ]:
```
visualize_q_values(q_values_intra, "incar_dest_B", 'Intra', 4, 3)
```



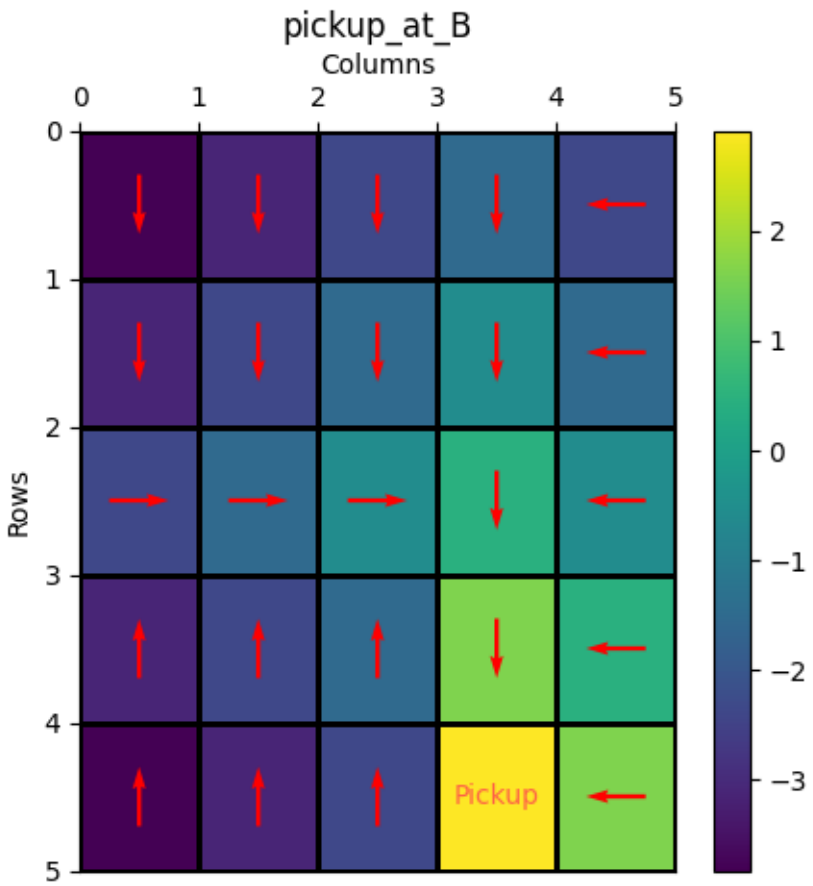
[ ]:
```
visualize_q_values(q_values_intra, "pickup_at_R", 'Intra', 0, 1)
```



[ ]:
```
visualize_q_values(q_values_intra, "pickup_at_Y", 'Intra', 2, 1)
```



[ ]:
```
visualize_q_values(q_values_intra, "pickup_at_B", 'Intra', 3, 2)
```



1 Comparing SMDP and Intra-option Q learning:

[ ]:
```
bestalpha = 0.4
bestepsilon = 0.15

q_values_smdp, uf_values_smdp, eps_rewards_smdp = SMDP_QLearning(bestalpha,␣
↪bestepsilon, 50000) # 30k to 50k
```

```
100%|      | 50000/50000 [00:51<00:00, 973.31it/s]
```

[ ]:
```
bestalpha = 0.4
bestepsilon = 0.15

q_values_intra, uf_values_intra, eps_rewards_intra =␣
↪IntraOption_QLearning(bestalpha, bestepsilon, 50000) # 30k to 50k
```

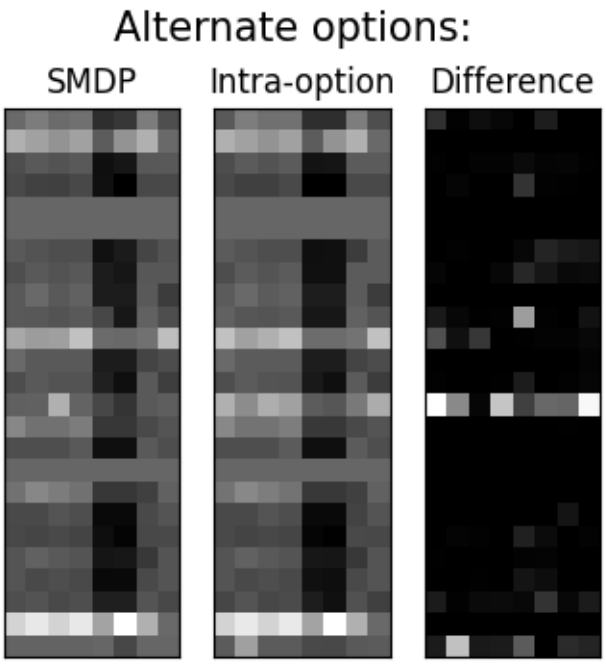
```
100%|      | 50000/50000 [00:45<00:00, 1087.57it/s]
```

[ ]:
```
def plotter(image, transf_image,diff, title_1, title_2):
    fig, (ax1, ax2,ax3) = plt.subplots(1, 3, figsize=(4,4))
    fig. suptitle("Alternate options:", fontsize=15)
    ax1.set_xticks([])
    ax1.set_yticks([])
    ax2.set_xticks([])
    ax2.set_yticks([])
    ax3.set_xticks([])
    ax3.set_yticks([])
    ax1.imshow(image, 'gray')
    ax2.imshow(transf_image, 'gray')
    ax1.title.set_text(title_1)
    ax2.title.set_text(title_2)
    ax3.imshow(diff, 'gray')
    ax3.title.set_text("Difference")
    plt.show()
```

1.1 Q Value plots

[ ]:
```
rng = np.random.default_rng()
idx = rng.choice(np.arange(len(q_values_smdp)), 25, replace=False)
```

[ ]:
```
image1=q_values_smdp[idx]
#image1 = rng.choice(image1, 50)
image2=q_values_intra[idx]
#image2 = rng.choice(image2, 50)AC
diff=np.abs(image1-image2)
plotter(image1,image2,diff,"SMDP","Intra-option")
```



1.1.1 Update frequency:

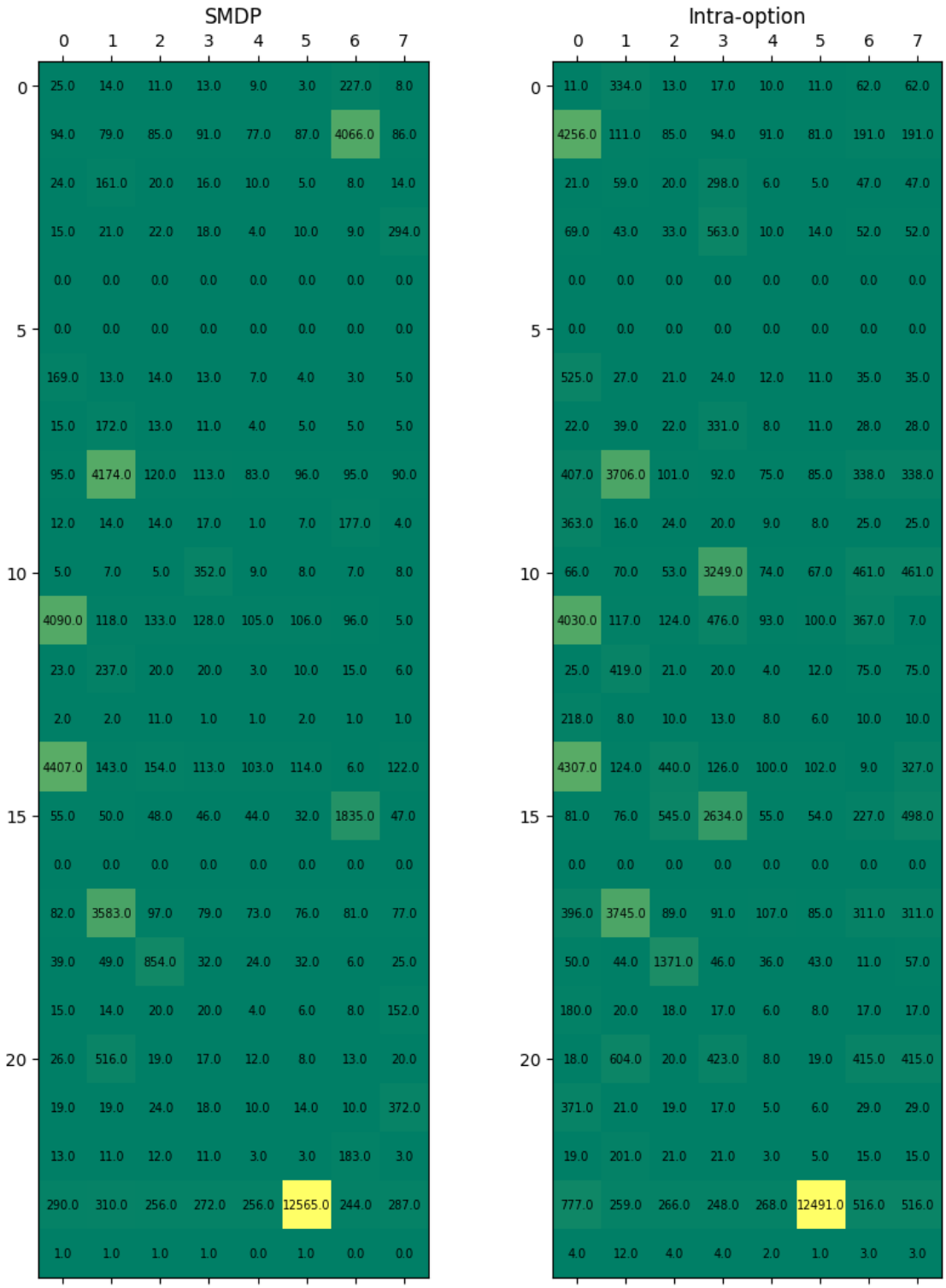
[ ]:
```
print("Total number of updates in SMDP: "+str(np.sum(uf_values_smdp)))
print("Total number of updates in Intra-option: "+str(np.sum(uf_values_intra)))
```

```
Total number of updates in SMDP: 586072.0
Total number of updates in Intra-option: 998700.0
```

[ ]:
```
def mat_plotter(image, transf_image):
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,13))
    fig. suptitle("Alternate options:", fontsize=15)
    ax1.matshow(image,cmap="summer")
    for (i, j), z in np.ndenumerate(image):
      ax1.text(j, i, '{:0.1f}'.format(z), ha='center',␣
 ↪va='center',fontsize="x-small")
    ax2.matshow(transf_image,cmap="summer")
    for (i, j), z in np.ndenumerate(transf_image):
      ax2.text(j, i, '{:0.1f}'.format(z), ha='center',␣
 ↪va='center',fontsize="x-small")
    ax1.title.set_text("SMDP")
    ax2.title.set_text("Intra-option")
    plt.show()
```

[ ]:
```
mat_plotter(uf_values_smdp[idx],uf_values_intra[idx])
```

Alternate options:



[ ]: