HW 4

Problem 1)

1) $G_i^s = \sum_{t=0}^{|\tau_i^s|} \gamma^t R(S_{t,i}^s, A_{t,i}^s) \leq \sum_{t=0}^{|\tau_i^s|} \gamma^t R_{max} = \frac{1-\gamma^\tau}{1-\gamma} R_{max}$

Let $\alpha = -\frac{1-\gamma^\tau}{1-\gamma} R_{max}$ and $\beta = \frac{1-\gamma^\tau}{1-\gamma} R_{max}$

Such that $\alpha \leq G_i \leq \beta$

2) $\mathbb{E}[E(s)] = \mathbb{E}\left[\sum_{i=1}^{N^s} G_i^s\right] = \sum_{i=1}^{N^s} \mathbb{E}[G_i^s] = \sum_{i=1}^{N^s} V^\pi(s) = N^s V^\pi(s)$

3) $\mathbb{P}(|E(s) - \mathbb{E}[E(s)]| \geq \varepsilon) \leq 2\exp\left(-\frac{2\varepsilon^2}{N^s(\beta-\alpha)^2}\right)$

where $\alpha = -\frac{1-\gamma^\tau}{1-\gamma} R_{max}$ $\beta = \frac{1-\gamma^\tau}{1-\gamma} R_{max}$

4) $V^\pi(s) = \frac{1}{N^s} E(s)$ $\varepsilon = N^s \varepsilon'$

$\mathbb{P}(|V^\pi(s) - \mathbb{E}[V^\pi(s)]| \geq \varepsilon') \leq 2\exp\left(-\frac{2N^s\varepsilon'}{(\beta-\alpha)^2}\right)$

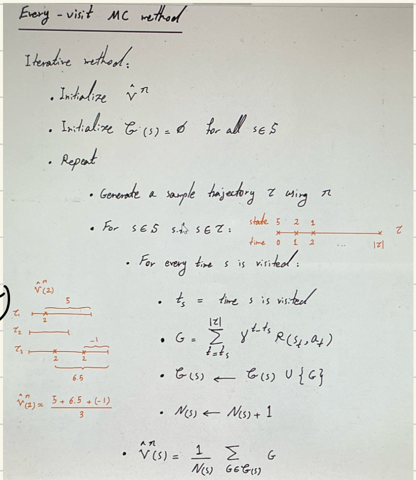
5) $\|\hat{V}^\pi(s) - V^\pi(s)\|_\infty \geq \varepsilon' \iff \bigcup_{s\in S} (|\hat{V}^\pi(s) - V^\pi(s)| \geq \varepsilon')$

$\therefore \mathbb{P}(\|\hat{V}^\pi(s) - V^\pi(s)\|_\infty \geq \varepsilon') \leq \sum_{s\in S} 2\exp\left(-\frac{2N^s\varepsilon'^2}{(\beta-\alpha)^2}\right)$

$2\exp\left(-\frac{2N^s\varepsilon'^2}{(\beta-\alpha)^2}\right) \leq 2\exp\left(-\frac{2N\varepsilon'^2}{(\beta-\alpha)^2}\right)$ where $N = \min_{s\in S} N^s$

$\therefore \mathbb{P}(\|\hat{V}^\pi(s) - V^\pi(s)\|_\infty \geq \varepsilon') \leq \sum_{s\in S} 2\exp\left(-\frac{2N\varepsilon'^2}{(\beta-\alpha)^2}\right)$

$$Q^{\pi}(s,x) = \frac{1}{N(s)} \sum_{G \in G(s)} G$$



Problem 2

τ_1: $G_1^1 = \gamma^0(-1) + \gamma^1(2)$ $\quad N=1$

$Q = -1 + \gamma 2$

τ_2: $G_2^1 = \gamma^0(-1) + \gamma(2) + \gamma^2(0) + \gamma^3(-1) + \gamma^4(1.5)$
$+ \gamma^5(1.5)$ $\quad N=2$

$$Q = \frac{G_1^1 + G_2^1}{N} = \frac{-1+\gamma^2-1+\gamma 2-\gamma^3+\gamma^4(1.5)+\gamma^5(1.5)}{2}$$

$G_2^2 = -1 + \gamma(1.5) + \gamma^2(1.5)$ $\quad N=3$

$$Q = \frac{G_1^1 + G_2^1 + G_2^2}{N} = \frac{-1+\gamma^2-1+\gamma 2-\gamma^3+\gamma^4(1.5)+\gamma^5(1.5)-1+\gamma(1.5)+\gamma^2(1.5)}{3}$$

τ_3: $G_3^1 = 0$ $\quad N=3$ $\quad Q=$

Problem 3

1) Each state except terminal states are run multiple times and an average is aggregated to get a more accurate estimate
2) States are randomly selected until all non-terminal states have seen visited >= 50 times

```python
import random
import matplotlib.pyplot as plt
import numpy as np

grid = [[0,0,0,0,1],
        [0,None,-1,0,0],
        [0,0,0,0,0],
        [0,0,None,None,0],
        [0,0,0,0,0]]
transitions = [["right", "right", "right", "right", "up"],
                ["left", "up", "left", "left", "up"],
                ["up", "up", "right", "right", "right"],
                ["up", "down", "down", "down", "up"],
                ["up", "right", "right", "up", "up"]]
discount_factor = 0.95
lr = 0.1
ACTIONS = ['up','down','left','right']

# get value function of each state in a trajectory
def getValue(trajectory):
    values = [[ [0, 0] for _ in range(5)] for _ in range(5)] # initialize all states to 0
    seen = set() # set of all seen states for first visit

    for i, step in enumerate(trajectory):
        state, reward, action = step
        if i == len(trajectory)-1: # don't count terminal states
            break
        if state in seen:
            continue
        else:
            r,c = state
            # since all rewards are 0 expect terminal state reward
            # G = gamma^(terminal-i) * reward at terminal
            values[r][c][0] += discount_factor**(len(trajectory) - 1 - i) * trajectory[-1][1]
            values[r][c][1] += 1
            seen.add(state)
    return values


def generateTrajectory(startrow, startcol): # keep track of all actions and states throughout trajectory
    trajectory = []
    r = startrow
    c = startcol
    reward = 0

    while reward == 0:
        # update reward
        reward = grid[r][c]
        # Random movement selection with weighted probabilities
        upProb = 0.85 if transitions[r][c] == "up" else 0.05
        downProb = 0.85 if transitions[r][c] == "down" else 0.05
        leftProb = 0.85 if transitions[r][c] == "left" else 0.05
        rightProb = 0.85 if transitions[r][c] == "right" else 0.05
        directions = ['left', 'right', 'up', 'down']
        probabilities = [leftProb, rightProb, upProb, downProb]

        if reward != 0:
            direction = 'terminate'
        else:
            direction = random.choices(directions, weights=probabilities)[0]

        # append to trajectory: state, reward, action
        if reward == None:
            trajectory.append(((r,c), 0, direction))
        else:
            trajectory.append(((r,c), reward, direction))

        # Update position based on random direction
        if direction == 'left' and c > 0 and grid[r][c-1] is not None:
            c -= 1
        elif direction == 'right' and c < 4 and grid[r][c+1] is not None:
            c += 1
        elif direction == 'up' and r > 0 and grid[r-1][c] is not None:
            r -= 1
        elif direction == 'down' and r < 4 and grid[r+1][c] is not None:
            r += 1
        # If movement is invalid, stay in current position

    return trajectory

def td0(startrow, startcol, value, visitCount):
    r = startrow
    c = startcol
    reward = 0

    while reward == 0:
        # Random movement selection with weighted probabilities
        upProb = 0.85 if transitions[r][c] == "up" else 0.05
        downProb = 0.85 if transitions[r][c] == "down" else 0.05
        leftProb = 0.85 if transitions[r][c] == "left" else 0.05
        rightProb = 0.85 if transitions[r][c] == "right" else 0.05
        directions = ['left', 'right', 'up', 'down']
        probabilities = [leftProb, rightProb, upProb, downProb]

        direction = random.choices(directions, weights=probabilities)[0]

        # Update position based on random direction
        nr,nc = r,c
        if direction == 'left' and c > 0 and grid[r][c-1] is not None:
            nc -= 1
        elif direction == 'right' and c < 4 and grid[r][c+1] is not None:
            nc += 1
        elif direction == 'up' and r > 0 and grid[r-1][c] is not None:
            nr -= 1
        elif direction == 'down' and r < 4 and grid[r+1][c] is not None:
            nr += 1
        # If movement is invalid, stay in current position

        # update value function for that state
        reward = grid[nr][nc]
        value[r][c] += lr * (reward + discount_factor*value[nr][nc] - value[r][c])
        visitCount[r][c] += 1
        r,c = nr,nc

def checkNumVisits(visitCount, minVisits):
    for r in range(5):
        for c in range(5):
            if grid[r][c] == 0 and visitCount[r][c] < minVisits:
                return False
    return True

def getReward():
    reward = [[0]*5 for _ in range(5)]
```

```python
    for r in range(5):
        for c in range(5):
            if grid[r][c] != 0:
                continue
            # Random movement selection with weighted probabilities
            upProb = 0.85 if transitions[r][c] == "up" else 0.05
            downProb = 0.85 if transitions[r][c] == "down" else 0.05
            leftProb = 0.85 if transitions[r][c] == "left" else 0.05
            rightProb = 0.85 if transitions[r][c] == "right" else 0.05

            if c > 0 and grid[r][c-1] is not None:
                reward[r][c] += leftProb * grid[r][c-1]
            else:
                reward[r][c] += leftProb * grid[r][c]
            if c < 4 and grid[r][c+1] is not None:
                reward[r][c] += rightProb * grid[r][c+1]
            else:
                reward[r][c] += rightProb * grid[r][c]
            if r > 0 and grid[r-1][c] is not None:
                reward[r][c] += upProb * grid[r-1][c]
            else:
                reward[r][c] += upProb * grid[r][c]
            if r < 4 and grid[r+1][c] is not None:
                reward[r][c] += downProb * grid[r+1][c]
            else:
                reward[r][c] += downProb * grid[r][c]

    return reward

def getTransition():
    # up down left right
    # rows = starting
    # column = ending
    P_pi = np.zeros((25,25))

    for r in range(5):
        for c in range(5):
            if grid[r][c] != 0:
                continue
            # Random movement selection with weighted probabilities
            upProb = 0.85 if transitions[r][c] == "up" else 0.05
            downProb = 0.85 if transitions[r][c] == "down" else 0.05
            leftProb = 0.85 if transitions[r][c] == "left" else 0.05
            rightProb = 0.85 if transitions[r][c] == "right" else 0.05

            if c > 0 and grid[r][c-1] is not None:
                P_pi[r*5 + c][r*5 + c-1] = leftProb
            else:
                P_pi[r*5 + c][r*5 + c] += leftProb
            if c < 4 and grid[r][c+1] is not None:
                P_pi[r*5 + c][r*5 + c+1] = rightProb
            else:
                P_pi[r*5 + c][r*5 + c] += rightProb
            if r > 0 and grid[r-1][c] is not None:
                P_pi[r*5 + c][(r-1)*5 + c] = upProb
            else:
                P_pi[r*5 + c][r*5 + c] += upProb
            if r < 4 and grid[r+1][c] is not None:
                P_pi[r*5 + c][(r+1)*5 + c] = downProb
            else:
                P_pi[r*5 + c][r*5 + c] += downProb

    return P_pi

if __name__ == '__main__':
    print("part 1")
    valueAggregate = [[[0, 0] for _ in range(5)] for _ in range(5)]
    for r in range(5):
        for c in range(5):
            for sample in range(3):
                trajectory = generateTrajectory(r,c)
                values = getValue(trajectory)
                # loop through all states in the trajectory to update total
                for trajr in range(5):
                    for trajc in range(5):
                        g, n = values[trajr][trajc]
                        if n != 0:
                            valueAggregate[trajr][trajc][0] += g
                            valueAggregate[trajr][trajc][1] += n
    # aggregate all values
    valueFunction = [[0]*5 for _ in range(5)]
    for r in range(5):
        for c in range(5):
            g,n = valueAggregate[r][c]
            valueFunction[r][c] = g / n if n > 0 else 0
    print(valueFunction)

    print("part 2")
    value = [[0]*5 for _ in range(5)]
    visitCount = [[0]*5 for _ in range(5)]

    while not checkNumVisits(visitCount, 50):
        row = random.randrange(0,5)
        col = random.randrange(0,5)
        while(grid[row][col] != 0):
            row = random.randrange(0,5)
            col = random.randrange(0,5)

        td0(row, col, value, visitCount)
    print(value)

    print("part 3")
    reward = getReward()
    P_pi = getTransition()
    r_vec = np.array(reward).flatten()
    V_N = np.linalg.inv(np.eye(25) - discount_factor * P_pi) @ r_vec
    print(V_N)

    print("part4")
    reward = getReward()
    P_pi = getTransition()
    r_vec = np.array(reward).flatten()
    v_true = np.linalg.inv(np.eye(25) - discount_factor * P_pi) @ r_vec

    tdValue = [[0]*5 for _ in range(5)]
    mcValue_helper = [[[0, 0] for _ in range(5)] for _ in range(5)]
    mcValue = [[0]*5 for _ in range(5)]
    visitCount = [[0]*5 for _ in range(5)]

    mc_error = []
    td_error = []
    episodes = 1000
```

```python
for i in range(episodes):
    row = random.randrange(0,5)
    col = random.randrange(0,5)
    while(grid[row][col] != 0):
        row = random.randrange(0,5)
        col = random.randrange(0,5)

    td0(row, col, tdValue, visitCount)

    trajectory = generateTrajectory(row, col)
    values = getValue(trajectory)
    # loop through all states in the trajectory to update total
    for trajr in range(5):
        for trajc in range(5):
            g, n = values[trajr][trajc]
            if n != 0:
                mcValue_helper[trajr][trajc][0] += g
                mcValue_helper[trajr][trajc][1] += n
    for r in range(5):
        for c in range(5):
            g,n = mcValue_helper[r][c]
            mcValue[r][c] = g / n if n > 0 else 0

    tderror = np.linalg.norm(np.array(tdValue).flatten() - v_true.flatten())
    mcerror = np.linalg.norm(np.array(mcValue).flatten() - v_true.flatten())
    td_error.append(tderror)
    mc_error.append(mcerror)

plt.plot(range(episodes), mc_error, label="Monte Carlo")
plt.plot(range(episodes), td_error, label="TD(0)")
plt.xlabel("Episodes")
plt.ylabel("Error")
plt.legend()
plt.grid(True)
plt.show()
```