Assignment 2

BY YUSEN ZHENG

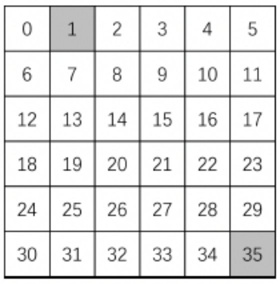
March 21, 2023

Email: zys0794@sjtu.edu.cn

Student ID: 520021911173

1 Introduction

In this report, we build a simple Gridworld. We implement both **first-visit and every-visit MC method and TD(0)** to evaluate an uniform random policy $\pi(n|\cdot)=\pi(e|\cdot)=\pi(s|\cdot)=\pi(w|\cdot)=0.25$. The gridworld is shown in the figure below.



2 Gridworld Environment

The Gridworld environment we use is basically the same as Assignment 1, and the constructor is as follows:

```
class Grid:
    def __init__(self, position, value=.0, is_terminal=False):
        self.val = value
        self.pos = position
        self.act = {'n': .0, 'e': .0, 's': .0, 'w': .0} if is_terminal else {
            'n': 0.25, 'e': 0.25, 's': 0.25, 'w': 0.25}
        self.is_terminal = is_terminal


class GridWorld:
    def __init__(self, width, height, terminal_list, gamma):
        self.w = width
        self.h = height
        self.terminal_list = terminal_list
        self.gamma = gamma
        self.r = -1
        self.grid_list = []
        for i in range(width*height):
            self.grid_list.append(Grid(i, is_terminal=i in terminal_list))
```

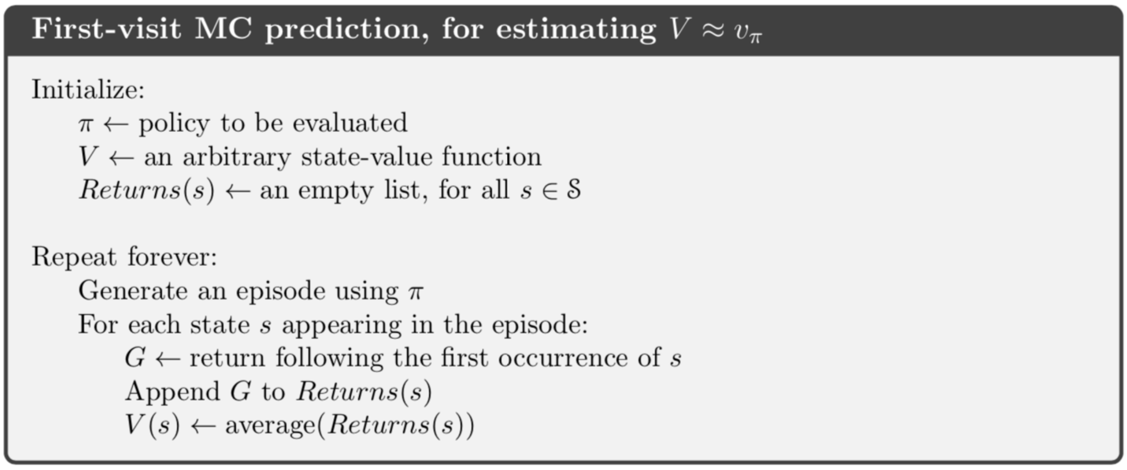
Because the three algorithms to be implemented in this assignment all need to sample a batch of episodes, we encapsulate the sampling of an episode into a function:

```
def generate_episode(self):
    episode = []
    start = random.randint(0, self.w*self.h-1)
    while self.grid_list[start].is_terminal:
        start = random.randint(0, self.w*self.h-1)
    episode.append(start)
    while True:
        if self.grid_list[start].is_terminal:
            break
        mov = random.choices(
            list(self.grid_list[start].act.keys()),
list(self.grid_list[start].act.values()))[0]
        if mov == 'n':
            start -= self.w if start >= self.w else 0
        elif mov == 'e':
            start += 1 if start % self.w != self.w-1 else 0
        elif mov == 's':
            start += self.w if start < self.w*(self.h-1) else 0
        elif mov == 'w':
            start -= 1 if start % self.w != 0 else 0
        episode.append(start)
    return episode
```

3 First-Visit Monte-Carlo Policy Evaluation

3.1 Algorithm

The original algorithm of First-Visit MC is as follows:



First-visit MC prediction, for estimating $V \approx v_\pi$

```
Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using $\pi$
    For each state $s$ appearing in the episode:
        $G \leftarrow$ return following the first occurrence of $s$
        Append $G$ to $Returns(s)$
        $V(s) \leftarrow$ average($Returns(s)$)
```

We use increment updates to save storage space:

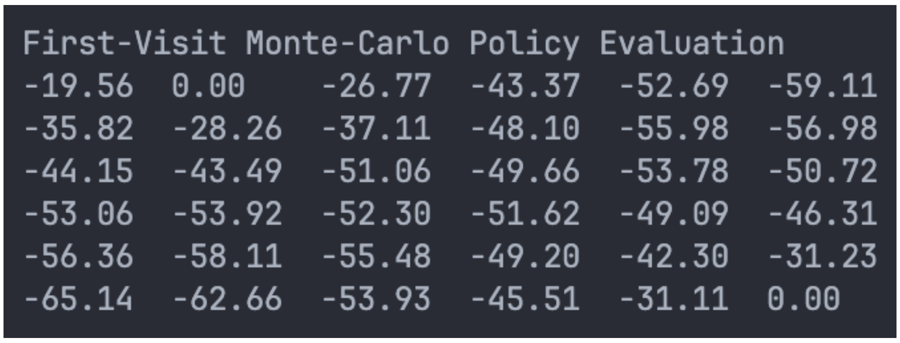
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

3.2 Implement

```
def first_visit_MC(self, episode_num=10000, alpha=.01):
    while episode_num:
        episode_num -= 1
        episode = self.generate_episode()
        G = 0
        for i in range(len(episode)-2, -1, -1):
            G = self.gamma*G+self.r
            if episode[i] not in episode[:i]:
                self.grid_list[episode[i]].val += alpha * \
                    (G-self.grid_list[episode[i]].val)
```

3.3 Result

In the experiment, we sample 10000 episodes, and the update step α is 0.01. The results of the first-visit MC policy evaluation are shown in the figure below:



```
First-Visit Monte-Carlo Policy Evaluation
-19.56  0.00    -26.77  -43.37  -52.69  -59.11
-35.82  -28.26  -37.11  -48.10  -55.98  -56.98
-44.15  -43.49  -51.06  -49.66  -53.78  -50.72
-53.06  -53.92  -52.30  -51.62  -49.09  -46.31
-56.36  -58.11  -55.48  -49.20  -42.30  -31.23
-65.14  -62.66  -53.93  -45.51  -31.11  0.00
```

4 Every-Visit Monte-Carlo Policy Evaluation

4.1 Algorithm

The algorithm of Every-Visit is very similar to that of First-Visit, the difference is as follows:

To evaluate state S,

- First-Visit only consider the first time-step t that state S is visited in an episode
- Every-Visit consider every time-step t that state S is visited in an episode

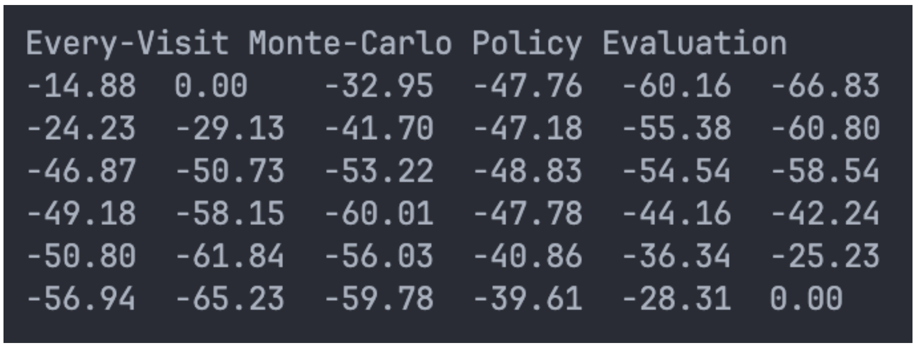
In Every-Visit Monte-Carlo Policy Evaluation, we also use increment updates.

4.2 Implement

```
def every_visit_MC(self, episode_num=10000, alpha=.01):
    while episode_num:
        episode_num -= 1
        episode = self.generate_episode()
        G = 0
        for i in range(len(episode)-2, -1, -1):
            G = self.gamma*G+self.r
            self.grid_list[episode[i]].val += alpha * \
                (G-self.grid_list[episode[i]].val)
```

4.3 Result

In the experiment, we sample 10000 episodes, and the update step α is 0.01. The results of the every-visit MC policy evaluation are shown in the figure below:

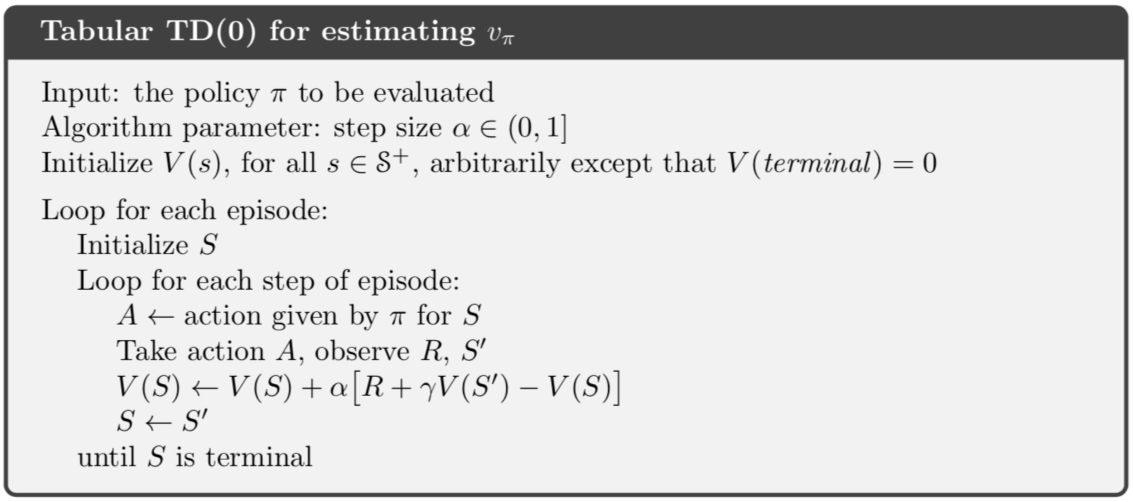


```
Every-Visit Monte-Carlo Policy Evaluation
-14.88  0.00    -32.95  -47.76  -60.16  -66.83
-24.23  -29.13  -41.70  -47.18  -55.38  -60.80
-46.87  -50.73  -53.22  -48.83  -54.54  -58.54
-49.18  -58.15  -60.01  -47.78  -44.16  -42.24
-50.80  -61.84  -56.03  -40.86  -36.34  -25.23
-56.94  -65.23  -59.78  -39.61  -28.31  0.00
```

5 Temporal-Difference Policy Evaluation

5.1 Algorithm

The algorithm of TD(0) is as follows:



Tabular TD(0) for estimating v_π

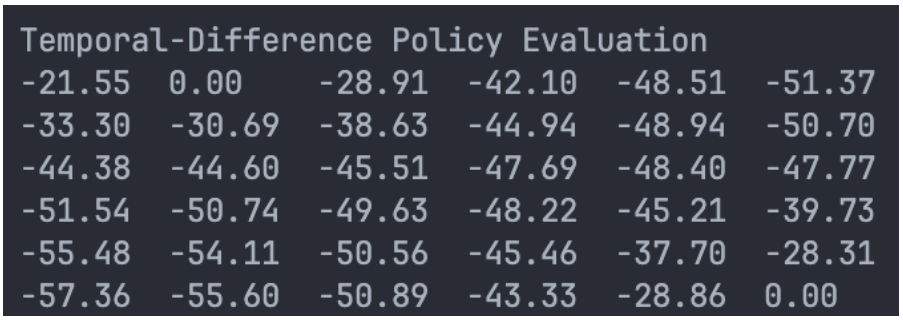
```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

5.2 Implement

```
def TD(self, episode_num=10000, alpha=.01):
    while episode_num:
        episode_num -= 1
        episode = self.generate_episode()
        for i in range(len(episode)-1):
            self.grid_list[episode[i]].val += alpha * \
                (self.r+self.gamma*self.grid_list[episode[i+1]].val -
                    self.grid_list[episode[i]].val)
```

5.3 Result

In the experiment, we sample 10000 episodes, and the update step α is 0.01. The results of the TD(0) policy evaluation are shown in the figure below:



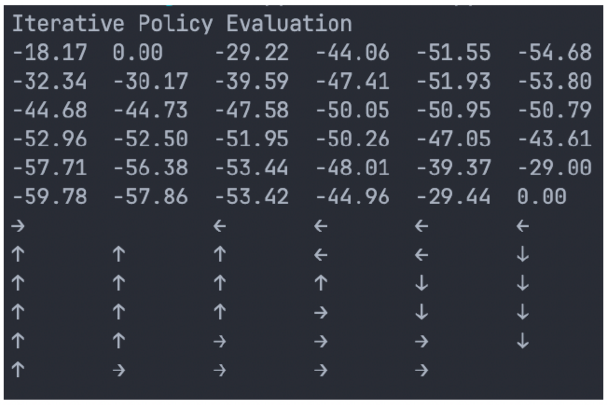
```
Temporal-Difference Policy Evaluation
-21.55  0.00    -28.91  -42.10  -48.51  -51.37
-33.30  -30.69  -38.63  -44.94  -48.94  -50.70
-44.38  -44.60  -45.51  -47.69  -48.40  -47.77
-51.54  -50.74  -49.63  -48.22  -45.21  -39.73
-55.48  -54.11  -50.56  -45.46  -37.70  -28.31
-57.36  -55.60  -50.89  -43.33  -28.86  0.00
```

6 Conclusion

Comparing the three policy evaluation algorithms, we found that the final converged state value functions are relatively close, and are close to the results obtained by using the iterative policy evaluation algorithm in Assignmnet 1:

3.3 Result

If agent follows uniform random policy $\pi(n|\cdot)=\pi(s|\cdot)=\pi(w|\cdot)=\pi(e|\cdot)=0.25$, the value function $v_\infty(s)$ and greedy policy w.r.t. $v_\infty(s)$ are as shown in the figure below.



```
Iterative Policy Evaluation
-18.17  0.00    -29.22  -44.06  -51.55  -54.68
-32.34  -30.17  -39.59  -47.41  -51.93  -53.80
-44.68  -44.73  -47.58  -50.05  -50.95  -50.79
-52.96  -52.50  -51.95  -50.26  -47.05  -43.61
-57.71  -56.38  -53.44  -48.01  -39.37  -29.00
-59.78  -57.86  -53.42  -44.96  -29.44  0.00
→               ←       ←       ←       ←
↑       ↑       ↑       ←       ←       ↓
↑       ↑       ↑       ↑       ↓       ↓
↑       ↑       ↑       →       ↓       ↓
↑       ↑       →       →       →       ↓
↑       →       →       →       →
```

We also tried different values of `episode_num` and `alpha`. The results show that the larger the number of episodes sampled, the more accurate the convergence result, while the time overhead will increase. When using a smaller alpha value, the change of the state value function is relatively stable, and at the same time it is easier to converge to the exact value. When using a larger alpha value, the jitter of the state value function is relatively strong.

A Source Code

```
import random


class Grid:
    def __init__(self, position, value=.0, is_terminal=False):
        self.val = value
        self.pos = position
        self.act = {’n’: .0, ’e’: .0, ’s’: .0, ’w’: .0} if is_terminal else {
            ’n’: 0.25, ’e’: 0.25, ’s’: 0.25, ’w’: 0.25}
        self.is_terminal = is_terminal


class GridWorld:
    def __init__(self, width, height, terminal_list, gamma):
        self.w = width
        self.h = height
        self.terminal_list = terminal_list
        self.gamma = gamma
        self.r = -1
        self.grid_list = []
        for i in range(width*height):
            self.grid_list.append(Grid(i, is_terminal=i in terminal_list))

    def print_val(self):
        for i in range(self.w*self.h):
            if i % self.w == 0 and i != 0:
                print(f’\n{self.grid_list[i].val:.2f}\t’, end=’’)
            elif i != self.w*self.h-1:
                print(f’{self.grid_list[i].val:.2f}\t’, end=’’)
            else:
                print(f'{self.grid_list[i].val:.2f}\t')

    def print_policy(self):
        arrows = ['↑', '→', '↓', '←']
        for i in range(self.w*self.h):
            mov = ''
            for j in range(4):
                if list(self.grid_list[i].act.items())[j][1] != .0:
                    mov += arrows[j]
            if i % self.w == 0 and i != 0:
                print(f'\n{mov}\t', end='')
            elif i != self.w*self.h-1:
                print(f'{mov}\t', end='')
            else:
                print(f'{mov}\t')

    def generate_episode(self):
        episode = []
        start = random.randint(0, self.w*self.h-1)
        while self.grid_list[start].is_terminal:
            start = random.randint(0, self.w*self.h-1)
        episode.append(start)
        while True:
            if self.grid_list[start].is_terminal:
                break
            mov = random.choices(
                list(self.grid_list[start].act.keys()),
list(self.grid_list[start].act.values()))[0]
            if mov == 'n':
                start -= self.w if start >= self.w else 0
            elif mov == 'e':
                start += 1 if start % self.w != self.w-1 else 0
            elif mov == 's':
                start += self.w if start < self.w*(self.h-1) else 0
            elif mov == 'w':
                start -= 1 if start % self.w != 0 else 0
            episode.append(start)
        return episode

    def first_visit_MC(self, episode_num=10000, alpha=.01):
        while episode_num:
            episode_num -= 1
            episode = self.generate_episode()
            G = 0
            for i in range(len(episode)-2, -1, -1):
                G = self.gamma*G+self.r
                if episode[i] not in episode[:i]:
                    self.grid_list[episode[i]].val += alpha * \
                        (G-self.grid_list[episode[i]].val)

    def every_visit_MC(self, episode_num=10000, alpha=.01):
        while episode_num:
            episode_num -= 1
            episode = self.generate_episode()
            G = 0
            for i in range(len(episode)-2, -1, -1):
                G = self.gamma*G+self.r
                self.grid_list[episode[i]].val += alpha * \
                    (G-self.grid_list[episode[i]].val)

    def TD(self, episode_num=10000, alpha=.01):
        while episode_num:
            episode_num -= 1
            episode = self.generate_episode()
            for i in range(len(episode)-1):
                self.grid_list[episode[i]].val += alpha * \
                    (self.r+self.gamma*self.grid_list[episode[i+1]].val -
                        self.grid_list[episode[i]].val)


if __name__ == '__main__':
    gw1 = GridWorld(6, 6, [1, 35], 1)
    print('\nFirst-Visit Monte-Carlo Policy Evaluation')
    gw1.first_visit_MC()
    gw1.print_val()
    gw2 = GridWorld(6, 6, [1, 35], 1)
    print('\nEvery-Visit Monte-Carlo Policy Evaluation')
    gw2.every_visit_MC()
    gw2.print_val()
    gw3 = GridWorld(6, 6, [1, 35], 1)
    print('\nTemporal-Difference Policy Evaluation')
    gw3.TD()
    gw3.print_val()
```