强化学习 实验一

姓名：吕奇澹

学号：SA23006171

1.实验目的

1. 理解、学习蒙特卡罗算法原理，并编码实现first-visit版本
2. 理解、学习TD算法原理，并编码实现SARSA、Q-learning

2.实验原理

特卡洛算法是一类通过模拟随机样本来解决问题的方法，通常可分为两类：

1. **问题具有内在的随机性：** 对于问题本身包含随机性的情况，利用计算机运算能力直接模拟这种随机过程。
2. **问题可转化为随机分布的特征数：** 将问题转化为某种随机分布的特征数，例如随机事件的概率或随机变量的期望值。通过随机抽样方法，估计随机事件发生的频率或估算随机变量的数字特征，并将其作为问题的解。

Sarsa算法：Sarsa算法是一种on-policy方法，其特点是原始策略和更新策略一致。不同于Monte Carlo方法的地方在于，Sarsa算法在执行完一个动作后就可以更新其值函数，而不需要采样一个完整的轨迹。这种方法适用于需要在每个时间步骤都更新值函数的情况。

Q-learning算法：Q-learning算法是一种off-policy方法。其原始策略和值函数更新策略不一致，而且不需要采样整个轨迹进行策略更新。不同于Sarsa算法的是，Q-learning在更新值函数时使用的是贪心策略而不是ϵ-greedy策略。这意味着在更新值函数时，Q-learning选择当前状态下最有价值的动作，而不是基于某个概率选择其他动作。

3.实验内容

Exp1 Monte-Carlo

Step1: Generate an episode:

```python
# step 1 : Generate an episode.
            # An episode is an array of (state, action, reward) tuples
        def generate_one_episode(env, generate_policy):
            trajectory = []
            state = env.reset()
            for i in range(1000):
                Pi_table = generate_policy(state)
                action = np.random.choice(np.arange(len(Pi_table)), p=Pi_table)
                next_state, reward, done, _ = env.step(action)
                trajectory.append((next_state, action, reward))
                if done:
                    break
                state = next_state
            return trajectory
        trajectory = generate_one_episode(env, policy)
```

Step 2 : Find all (state, action) pairs we've visited in this episode:

```python
s_a_pairs = set([(x[0], x[1]) for x in trajectory])
```

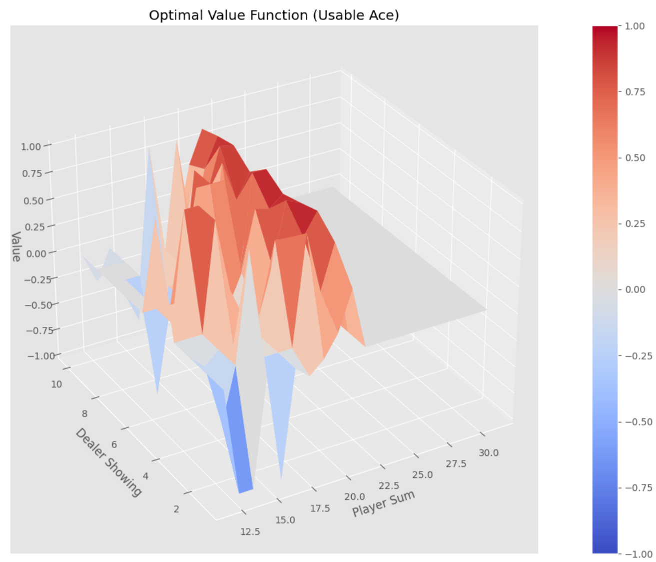
Step 3 : Calculate average return for this state over all sampled episodes:

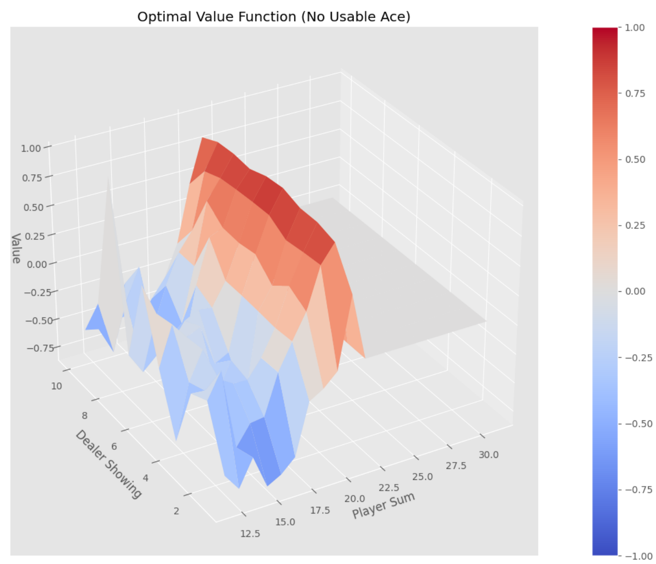
- First-visit version:

```python
for state, action in s_a_pairs:
    s_a = (state, action)
    first_visit_id = next(i for i, x in enumerate(trajectory) if x[0] == state and x[1] == action)
    G = sum([x[2] * (discount_factor ** i) for i, x in enumerate(trajectory[first_visit_id:])])
    returns_sum[s_a] += G
    returns_count[s_a] += 1.
    Q[state][action] = returns_sum[s_a] / returns_count[s_a]
```

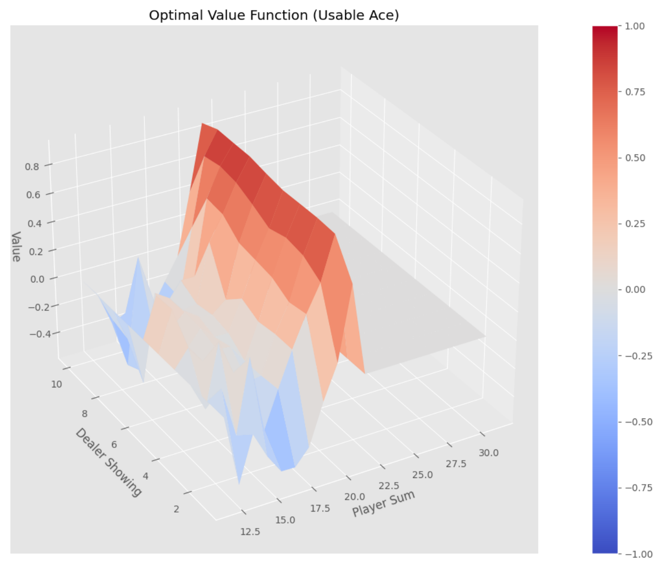
- Corresponding Results:

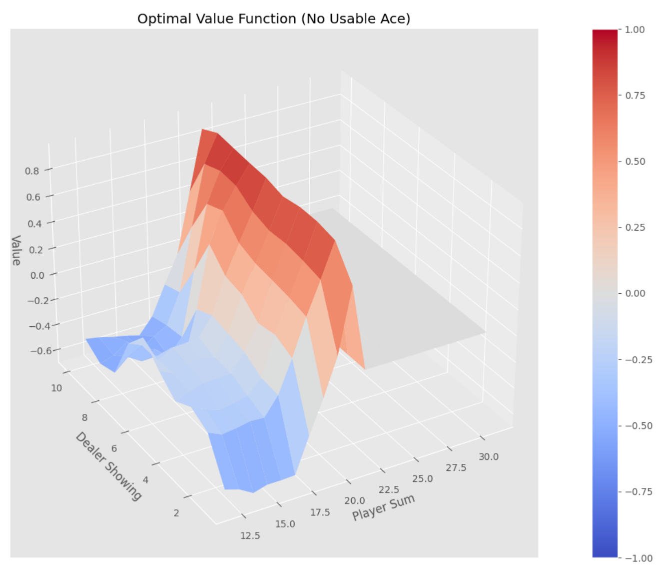
1. 10000 episodes:





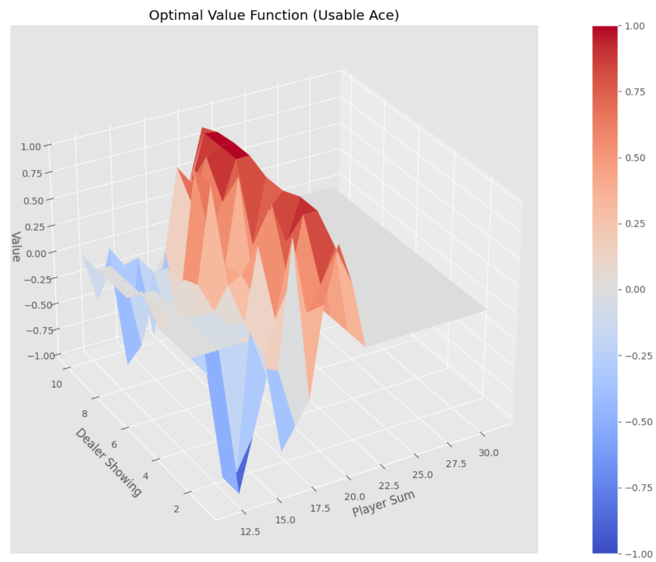
2. 500000 episodes:

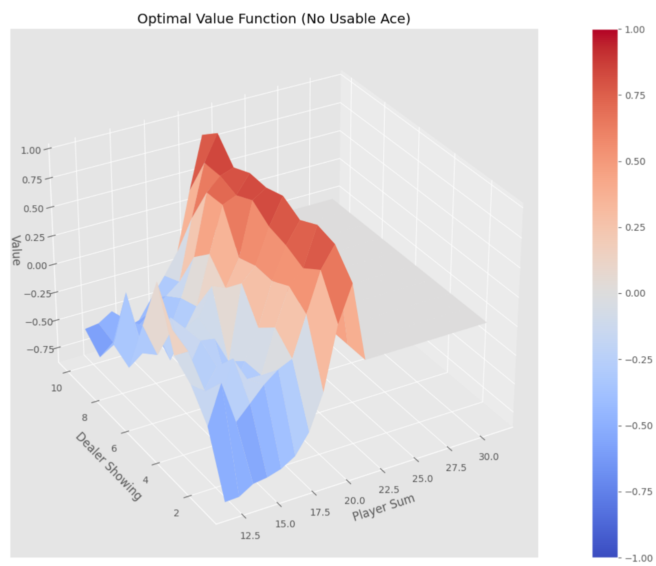




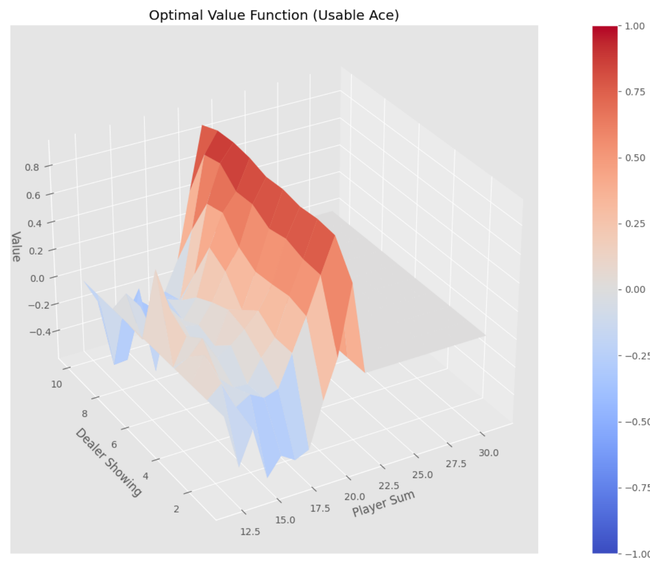
- Every-visit version:

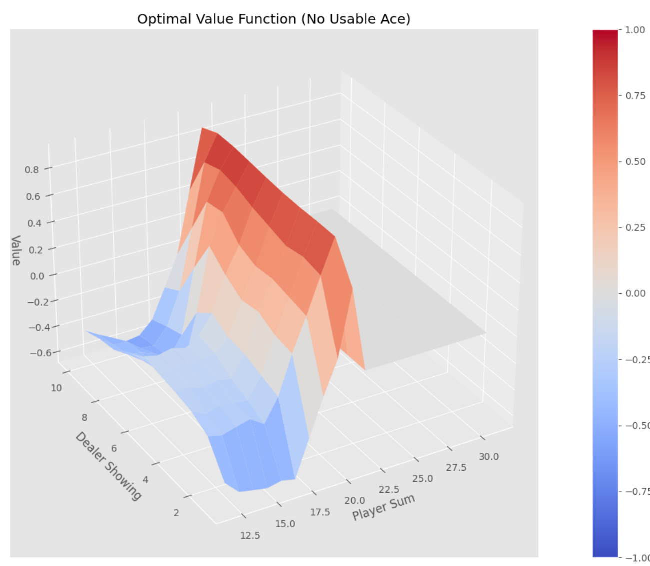
1. 10000 episodes:





2. 500000 episodes:





- **对比every-visit和first-visit**，可以发现它们的主要区别在于对重复出现的相同状态 计算对应的reward.**对于first-visit**，每种状态仅计算第一次出现的reward；**对于every-visit**，每种状态计算reward均值。

Exp2 Sarsa

step 1： Take a step(1 line code, tips : env.step()):

```python
new_state, reward, done, _ = env.step(action)
```

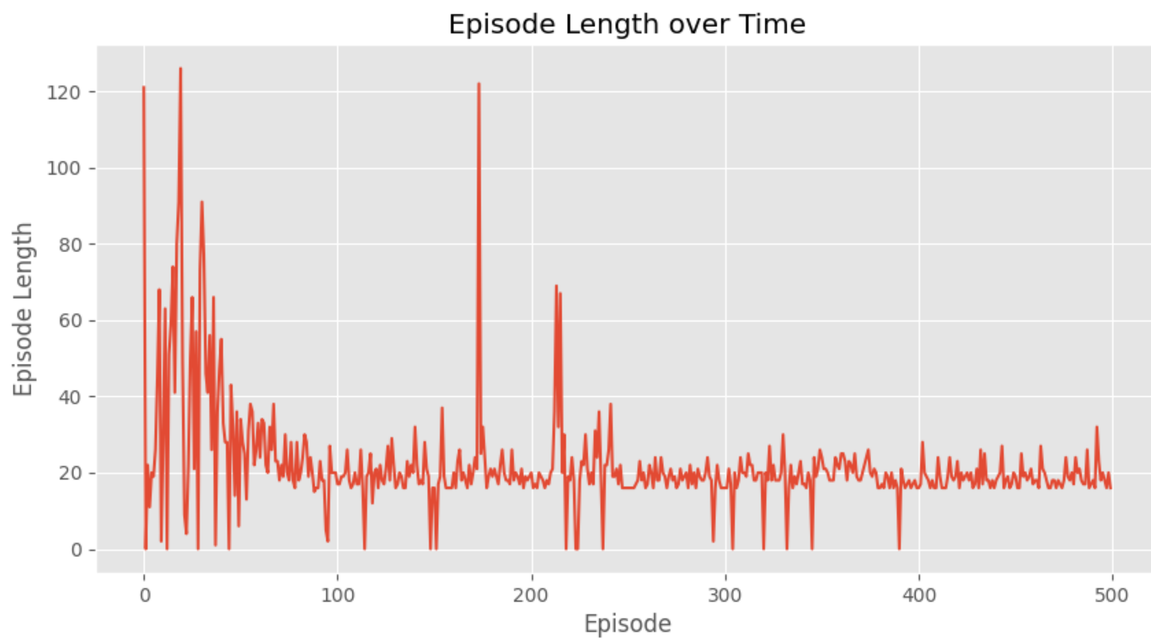
Step 2: Choose the next action:

```python
new_action = np.random.choice(np.arange(env.action_space.n), p=policy(new_state))
stats.episode_rewards[i_episode] += reward
stats.episode_lengths[i_episode] = t
```

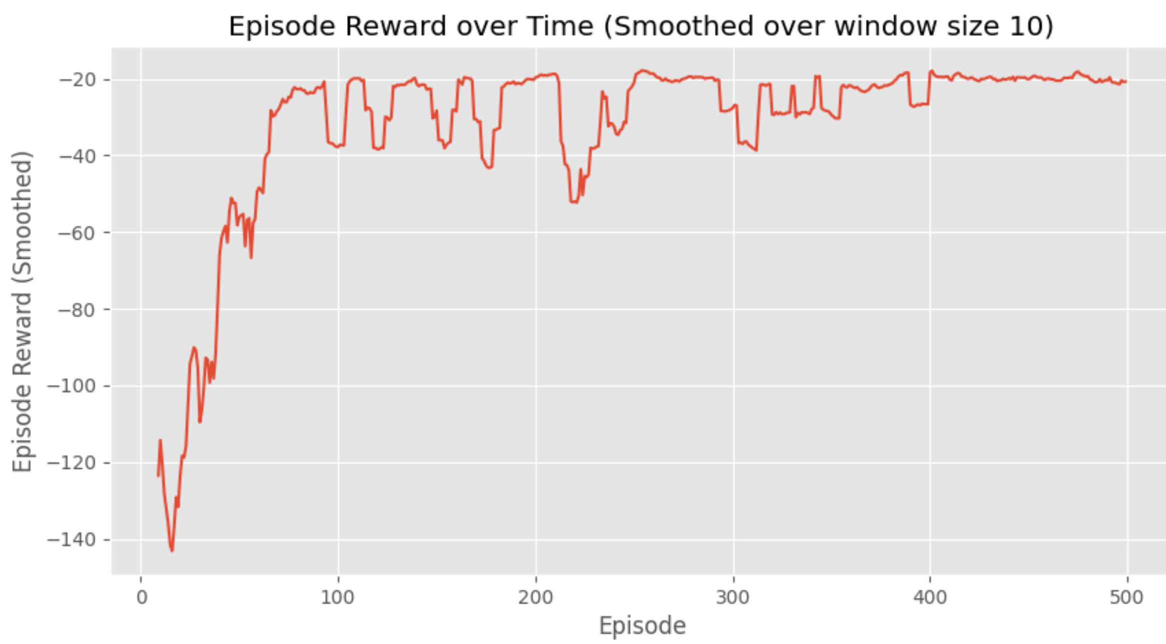
step 3： Pick the next action

```python
if done:
    Q[state][action] = Q[state][action] + alpha * (reward + discount_factor * 0.0 - Q[state][action])
    break
else:
    Q[state][action] = Q[state][action] + alpha * (reward + discount_factor * Q[new_state][new_action] - Q[state][action])
    state = new_state
    action = new_action
```

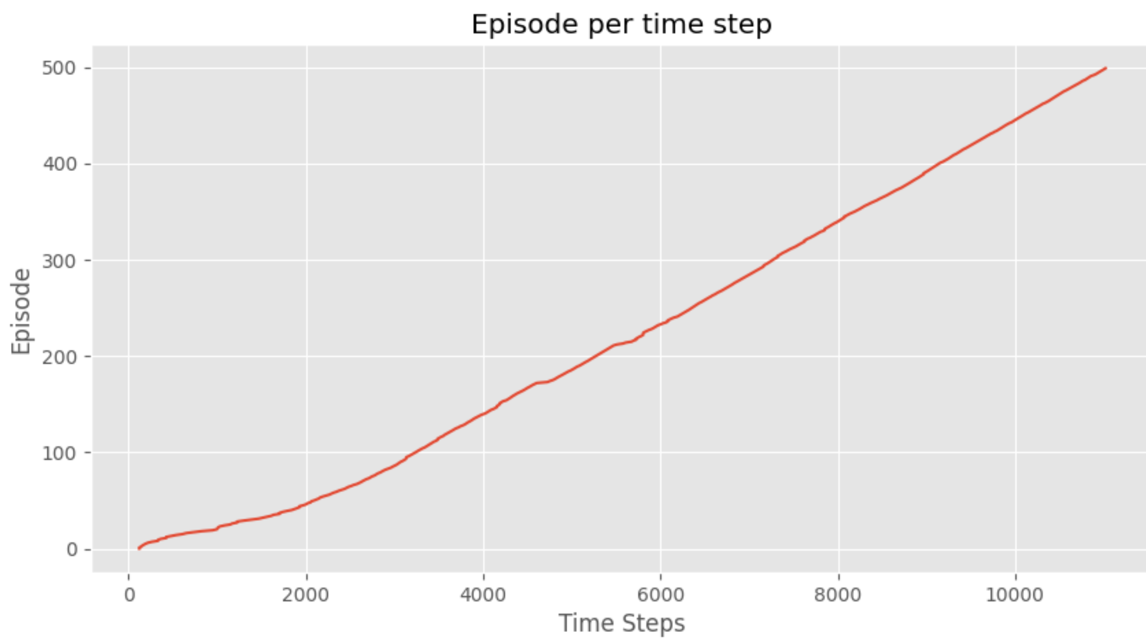
- Corresponding Results:



Episode Length over Time
Episode Length
Episode
0
20
40
60
80
100
120
0
100
200
300
400
500



Episode Reward over Time (Smoothed over window size 10)
Episode Reward (Smoothed)
Episode
−20
−40
−60
−80
−100
−120
−140
0
100
200
300
400
500



Exp3 Q-learning

1. 实现Q-learning 算法

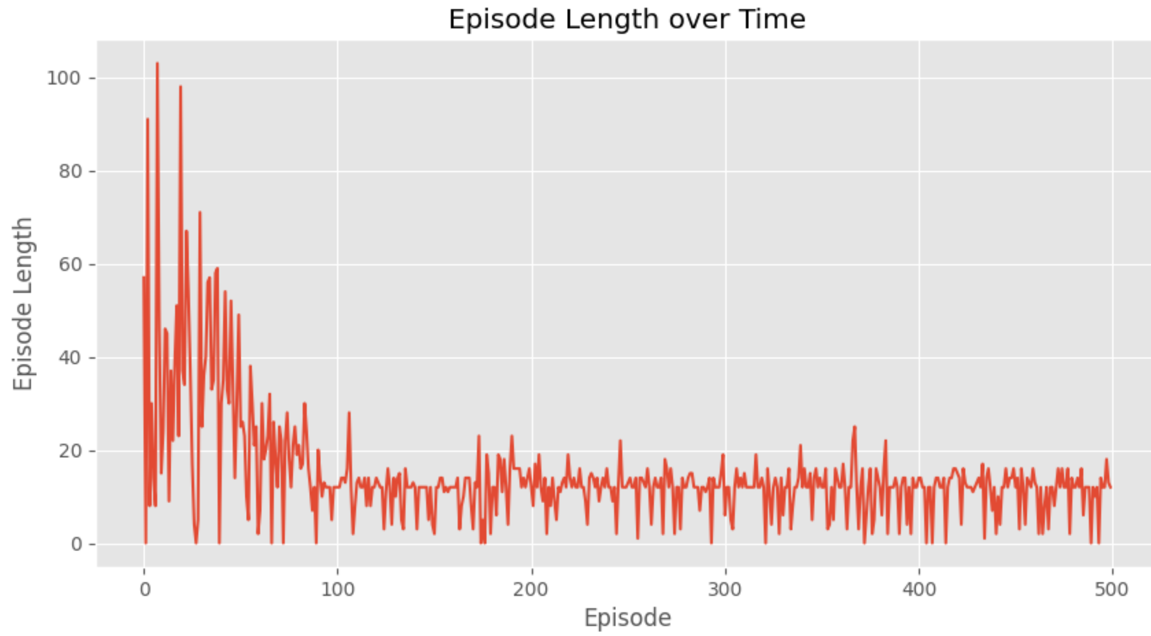
step 1 : Take a step:

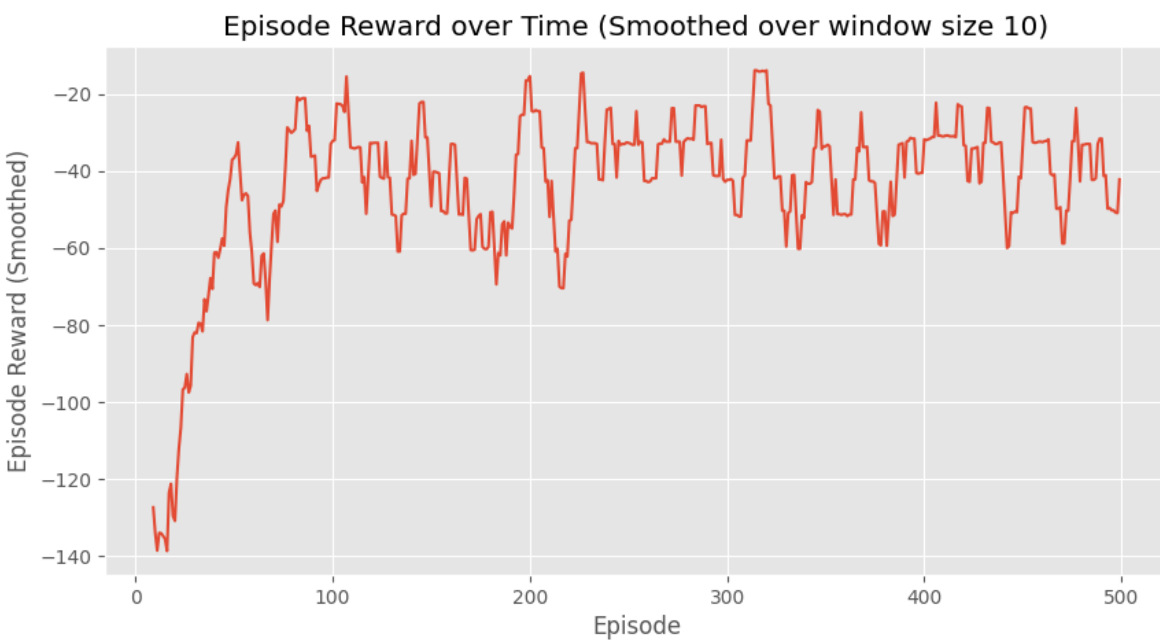
```python
action = np.random.choice(np.arange(env.action_space.n), p=double_q_policy(state))
new_state, reward, done, _ = env.step(action)
stats.episode_rewards[i_episode] += reward
stats.episode_lengths[i_episode] = t
```

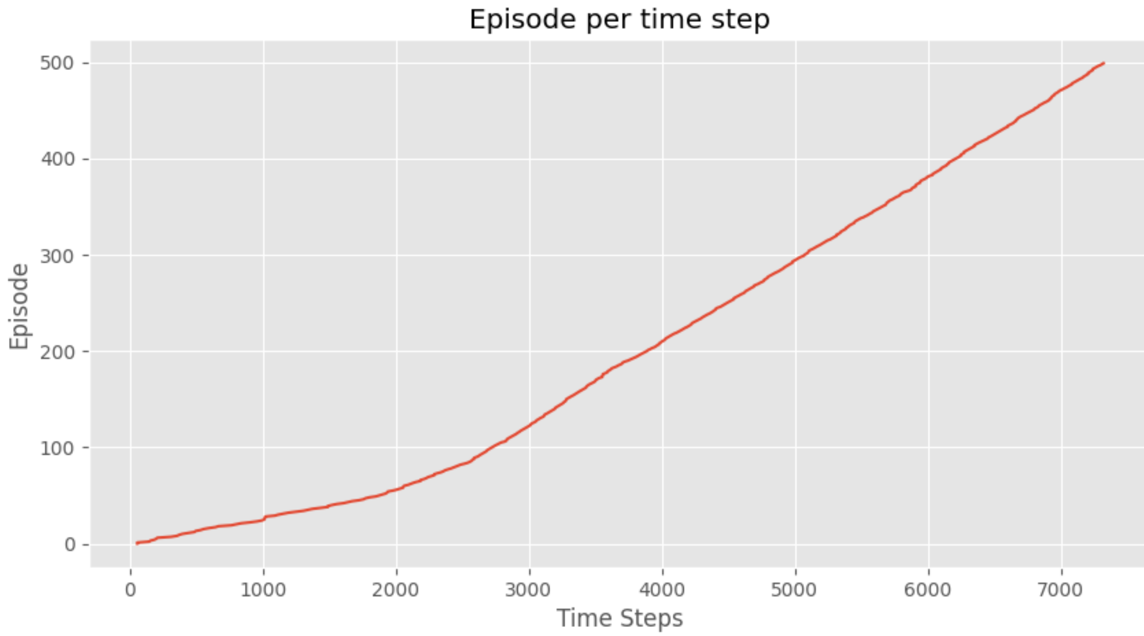
step 2 : TD Update:

```python
if random.random() >= 0.5:
    next_action = np.argmax(Q1[new_state])
    Q1[state][action] += alpha * (reward + discount_factor * Q2[new_state][next_action] - Q1[state][action])
else:
    next_action = np.argmax(Q2[new_state])
    Q2[state][action] += alpha * (reward + discount_factor * Q1[new_state][next_action] - Q2[state][action])
if done:
    break
state = new_state
```

- Corresponding Results：


Episode Length over Time
Episode Length
Episode
0
20
40
60
80
100
0
100
200
300
400
500


Episode Reward over Time (Smoothed over window size 10)
Episode Reward (Smoothed)
Episode
−20
−40
−60
−80
−100
−120
−140
0
100
200
300
400
500



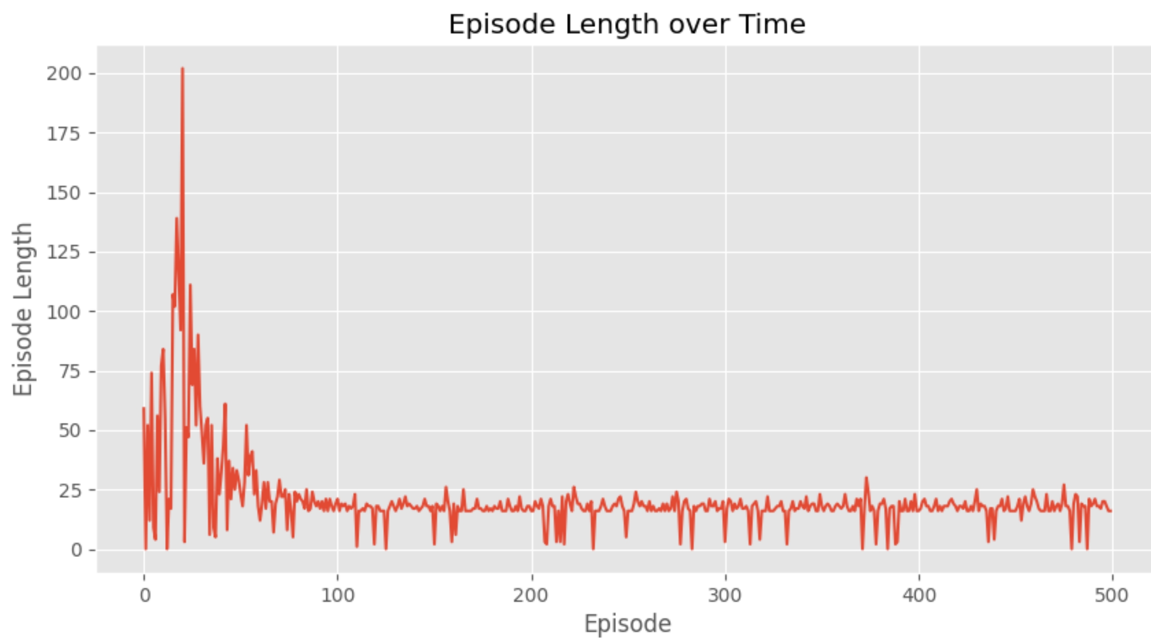
2. 实现double-Q learning算法

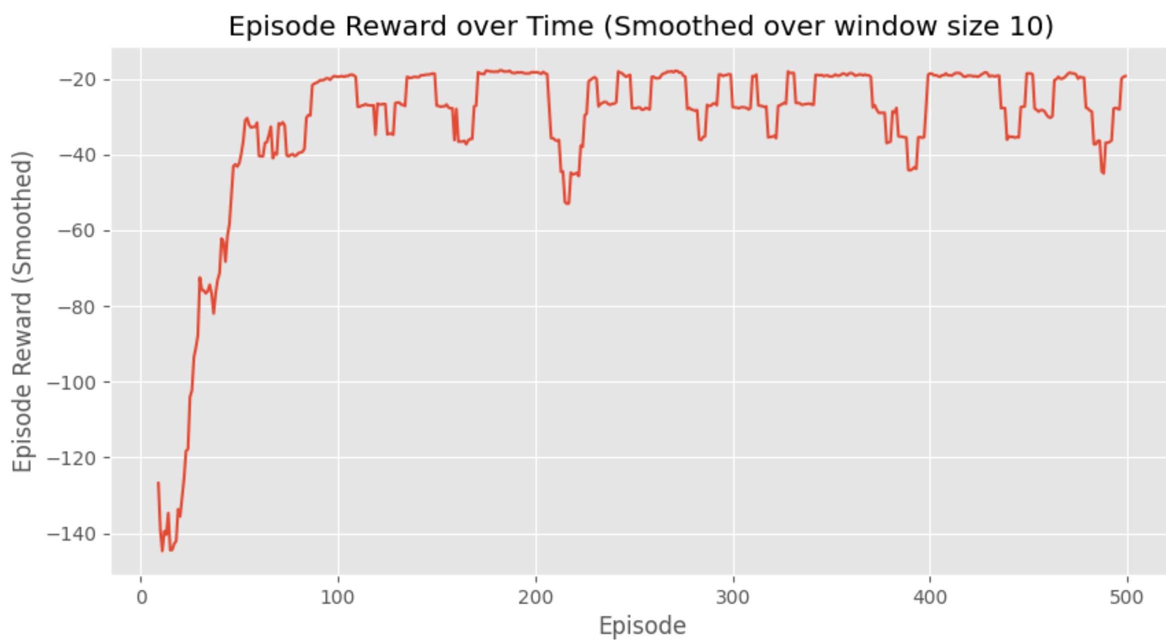
Main implementation codes:

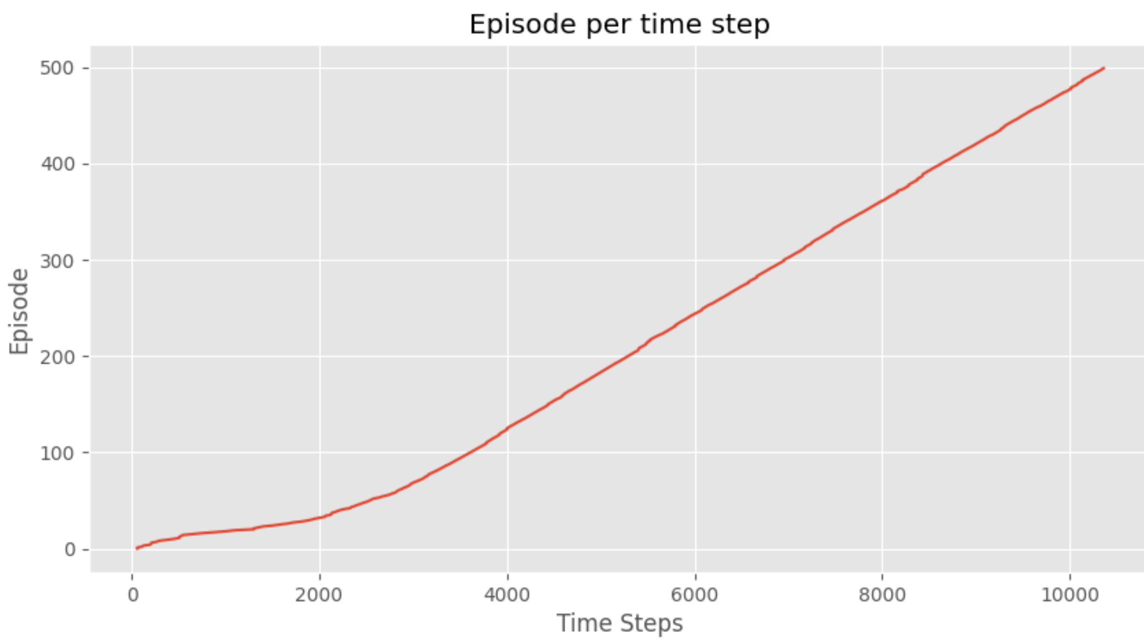
```python
def make_double_epsilon_greedy_policy(Q1, Q2, epsilon, nA):
    def policy_fn(observation):
        A = np.ones(nA, dtype=float) * epsilon / nA
        best_action = np.argmax(Q1[observation] + Q2[observation])
        A[best_action] += (1.0 - epsilon)
        return A
    return policy_fn


def double_q_learning(env, num_episodes, discount_factor=1.0, alpha=0.5, epsilon=0.1):


    # The final action-value function.
    # A nested dictionary that maps state -> (action -> action-value).
    Q1 = defaultdict(lambda: np.zeros(env.action_space.n))
    Q2 = defaultdict(lambda: np.zeros(env.action_space.n))

    # Keeps track of useful statistics
    stats = plotting.EpisodeStats(
        episode_lengths=np.zeros(num_episodes),
        episode_rewards=np.zeros(num_episodes))

    # The policy we're following
    double_q_policy = make_double_epsilon_greedy_policy(Q1, Q2, epsilon, env.action_space.n)

    for i_episode in range(num_episodes):
        # Print out which episode we're on, useful for debugging.
        if (i_episode + 1) % 100 == 0:
            print("\rEpisode {}/{}.".format(i_episode + 1, num_episodes), end="")
            sys.stdout.flush()

        # Reset the environment and pick the first action
        state = env.reset()
        for t in itertools.count():
 ########################################Implement your code here##########################################################################

            # step 1 : Take a step
            action = np.random.choice(np.arange(env.action_space.n), p=double_q_policy(state))
            new_state, reward, done, _ = env.step(action)
```

```

            # Update statistics
            stats.episode_rewards[i_episode] += reward
            stats.episode_lengths[i_episode] = t

            # step 2 : TD Update
            if random.random() >= 0.5:
                next_action = np.argmax(Q1[new_state])
                Q1[state][action] += alpha * (reward + discount_factor * Q2[new_state][next_action] - Q1[state][action])
            else:
                next_action = np.argmax(Q2[new_state])
                Q2[state][action] += alpha * (reward + discount_factor * Q1[new_state][next_action] - Q2[state][action])
            if done:
                break
            state = new_state

########################################Imlement your code end#############################################################################
    return Q1, Q2, stats
```

- Corresponding Results


Episode Length over Time
Episode Length
Episode
0
25
50
75
100
125
150
175
200
0
100
200
300
400
500


Episode Reward over Time (Smoothed over window size 10)
Episode Reward (Smoothed)
Episode
−20
−40
−60
−80
−100
−120
−140
0
100
200
300
400
500



- Comparison between Q–learning & double–Q learning: Q–learning在每次选择动作时总是选择具有最大Q值的动作，但如果该动作是一个噪声动作，或者奖励是随机值时，可能会导致最大化偏差。这个问题的一个原因是在优化时使用了相同的样本。为了解决这个问题，Double Q–learning使用了两组独立的样本进行估计。根据结果分析，Double Q–learning的奖励曲线通常比Q–learning的曲线更加稳定。