

PART 1

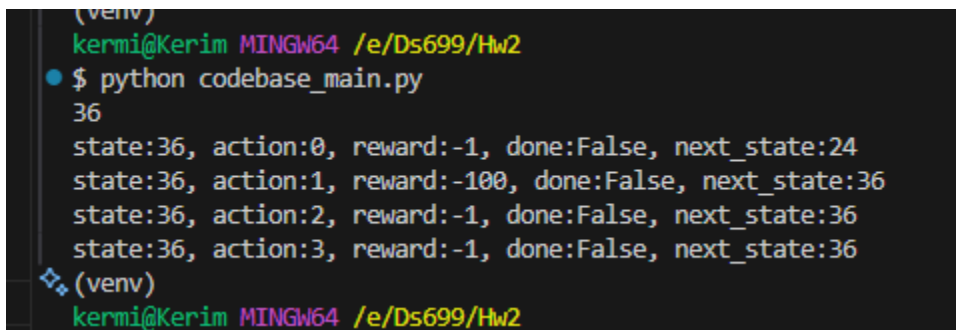
(a) You do not need to write any code in this question. Please:

- set the parameter `method` in `get_args.py` as `test-cliff-walking`.
- run `codebase_main.py` and the `test_each_action` function.

(1) Please take a screenshot of all the output results from the code.

(2) **Question:** What is the initial state number? Where is it located on the map?

(Hint: The location is represented as `[#row, #col]`, e.g., the top-left corner's location is represented as `[0,0]`.)



```
(venv)
kermi@Kerim MINGW64 /e/Ds699/Hw2
$ python codebase_main.py
36
state:36, action:0, reward:-1, done:False, next_state:24
state:36, action:1, reward:-100, done:False, next_state:36
state:36, action:2, reward:-1, done:False, next_state:36
state:36, action:3, reward:-1, done:False, next_state:36
(venv)
kermi@Kerim MINGW64 /e/Ds699/Hw2
```

The initial state number is 36. For the position we can see two ways 3 ways take you back to 36 while one takes you to 24. If correct, action 0 is up, action 1 is right, action 2 is down and action 3 is left. Which makes this bottom right corner since you can only go up and a cliff to your right.

(3) **Question:** What do the actions 0, 1, 2, and 3 each represent?

Actions: 0 = up, 1 = right, 2 = down, 3 = left

(4) **Question:** After taking action 0, which state does the agent reach? What is the reward?

When you go up the new state will be 20 and the reward will be -1

(5) **Question:** Which position do the state numbers 0, 11, and 47 represent? Can you briefly describe how the state number corresponds with the position?

Since the grid size is 4 by 12 we know that 0 is the top left corner since 36 is the bottom left corner. 11 is the top right, 47 is bottom right.

(6) **Question:** After taking action 1, which state does the agent reach? What is the reward? Is the agent terminated?

(Hint: The episode terminates when the player enters state [47] (location [3, 11]). You can check the definition of "Episode End" from the [official Gymnasium documentation](#).)

Action 1 will fall off the cliff and reset back to the initial position which with a reward of -100 and state of 36

(7) **Question:** After taking action 2 and 3, which state does the agent reach? What is the reward? Can you explain what does it mean?

After taking action 2 and 3 the agent will reach the state of 36 because it is at the edge of the grid. Since the agent will still be in the grid it will not get penalized like action 1 but will be a wasted action.

(8) **Question:** Can you briefly describe the transition and reward rules of Cliff Walking environment?

Transition and reward rules of the Cliff Walking Environment is like a person walking across a cliff, if the person is not careful they could potentially fall off but if they are successful they will make it across the cliff. The person has actions of up, right, left, and down and each safe move you make you get a reward of -1 but if you fall off you would be eliminated getting -100.

PART 2

(b) In this question, you need to finish your code in `codebase_cliff_walking_test.py` and `codebase_main.py` between the pound sign lines denoted with "Your Code" in each function, then run the program in `codebase_main.py` to test your code. First, you need to implement the function `test_moves` in the `codebase_cliff_walking_test.py` file.

- First, reset the environment.
- Then, take the action in the parameter `actions` one by one for each step.
- Print the `action`, `next state`, `reward`, and `done` for each step.

You need to call the `env.step` and `env.reset` to finish this function.

Then, in `codebase_main.py` file:

- Create a list of actions to direct the agent moving from the starting position to the most top-right corner position.
- Call `test_moves` function to test if your series of actions are correct.

(1) Please take a screenshot of your result. The screenshot must include the last state that your actions lead the environment to.

```
(venv)
kermi@Kerim MINGW64 /e/Ds699/Hw2
$ python codebase_main.py
36
state:36, action:0, reward:-1, done:False, next_state:24
state:36, action:1, reward:-100, done:False, next_state:36
state:36, action:2, reward:-1, done:False, next_state:36
state:36, action:3, reward:-1, done:False, next_state:36

--- Testing Move to Top-Right Corner ---
Initial State: 36
Step 1: Action=0, Next State=24, Reward=-1, Done=False
Step 2: Action=0, Next State=12, Reward=-1, Done=False
Step 3: Action=0, Next State=0, Reward=-1, Done=False
Step 4: Action=1, Next State=1, Reward=-1, Done=False
Step 5: Action=1, Next State=2, Reward=-1, Done=False
Step 6: Action=1, Next State=3, Reward=-1, Done=False
Step 7: Action=1, Next State=4, Reward=-1, Done=False
Step 8: Action=1, Next State=5, Reward=-1, Done=False
Step 9: Action=1, Next State=6, Reward=-1, Done=False
Step 10: Action=1, Next State=7, Reward=-1, Done=False
Step 11: Action=1, Next State=8, Reward=-1, Done=False
Step 12: Action=1, Next State=9, Reward=-1, Done=False
Step 13: Action=1, Next State=10, Reward=-1, Done=False
Step 14: Action=1, Next State=11, Reward=-1, Done=False
total reward:-14

--- Testing Optimal Path to Goal ---
Initial State: 36
Step 1: Action=0, Next State=24, Reward=-1, Done=False
Step 2: Action=1, Next State=25, Reward=-1, Done=False
Step 3: Action=1, Next State=26, Reward=-1, Done=False
Step 4: Action=1, Next State=27, Reward=-1, Done=False
Step 5: Action=1, Next State=28, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 8: Action=1, Next State=31, Reward=-1, Done=False
Step 9: Action=1, Next State=32, Reward=-1, Done=False
Step 10: Action=1, Next State=33, Reward=-1, Done=False
Step 11: Action=1, Next State=34, Reward=-1, Done=False
Step 12: Action=1, Next State=35, Reward=-1, Done=False
Step 13: Action=2, Next State=47, Reward=-1, Done=True
total reward:-13
❖ (venv)
kermi@Kerim MINGW64 /e/Ds699/Hw2
$
```

- (2) **Question:** What is the highest total reward for an episode in this environment?
(**Hint:** An episode must start from the initial state and end in a terminal state. The top-right corner state in this question is not terminal. Each time step incurs -1 reward, unless the player stepped into the cliff, which incurs -100 reward.)

The highest possible reward is -13

(c) In the `codebase_main.py` file, please:

- Create a list of actions to direct the agent moving from the initial state to the terminal state (the bottom-right corner of the map) and has the highest total reward.
 - Call the `test_moves` function to test if your series of actions are correct.
- (1) Please take a screenshot of your result. The screenshot must include the last state that your actions lead the environment to.

```
(venv)
```

```
kermi@Kerim MINGW64 /e/Ds699/Hw2
```

```
• $ python codebase_main.py
```

```
36
```

```
state:36, action:0, reward:-1, done:False, next_state:24
```

```
state:36, action:1, reward:-100, done:False, next_state:36
```

```
state:36, action:2, reward:-1, done:False, next_state:36
```

```
state:36, action:3, reward:-1, done:False, next_state:36
```

```
--- Testing Move to Top-Right Corner ---
```

```
Initial State: 36
```

```
Step 1: Action=0, Next State=24, Reward=-1, Done=False
```

```
Step 2: Action=0, Next State=12, Reward=-1, Done=False
```

```
Step 3: Action=0, Next State=0, Reward=-1, Done=False
```

```
Step 4: Action=1, Next State=1, Reward=-1, Done=False
```

```
Step 5: Action=1, Next State=2, Reward=-1, Done=False
```

```
Step 6: Action=1, Next State=3, Reward=-1, Done=False
```

```
Step 7: Action=1, Next State=4, Reward=-1, Done=False
```

```
Step 8: Action=1, Next State=5, Reward=-1, Done=False
```

```
Step 9: Action=1, Next State=6, Reward=-1, Done=False
```

```
Step 10: Action=1, Next State=7, Reward=-1, Done=False
```

```
Step 11: Action=1, Next State=8, Reward=-1, Done=False
```

```
Step 12: Action=1, Next State=9, Reward=-1, Done=False
```

```
Step 13: Action=1, Next State=10, Reward=-1, Done=False
```

```
Step 14: Action=1, Next State=11, Reward=-1, Done=False
```

```
total reward:-14
```

```
--- Testing Optimal Path to Goal ---
```

```
Initial State: 36
```

```
Step 1: Action=0, Next State=24, Reward=-1, Done=False
```

```
Step 2: Action=1, Next State=25, Reward=-1, Done=False
```

```
Step 3: Action=1, Next State=26, Reward=-1, Done=False
```

```
Step 4: Action=1, Next State=27, Reward=-1, Done=False
```

```
Step 5: Action=1, Next State=28, Reward=-1, Done=False
```

```
Step 6: Action=1, Next State=29, Reward=-1, Done=False
```

```
Step 7: Action=1, Next State=30, Reward=-1, Done=False
```

```
Step 8: Action=1, Next State=31, Reward=-1, Done=False
```

```
Step 9: Action=1, Next State=32, Reward=-1, Done=False
```

```
Step 10: Action=1, Next State=33, Reward=-1, Done=False
```

```
Step 11: Action=1, Next State=34, Reward=-1, Done=False
```

```
Step 12: Action=1, Next State=35, Reward=-1, Done=False
```

```
Step 13: Action=2, Next State=47, Reward=-1, Done=True
```

```
total reward:-13
```

```
--- Testing Highest Reward Path to Terminal State ---
Initial State: 36
Step 1: Action=0, Next State=24, Reward=-1, Done=False
Step 2: Action=1, Next State=25, Reward=-1, Done=False
Step 3: Action=1, Next State=26, Reward=-1, Done=False
Step 4: Action=1, Next State=27, Reward=-1, Done=False
Step 5: Action=1, Next State=28, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 8: Action=1, Next State=31, Reward=-1, Done=False
Step 9: Action=1, Next State=32, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 8: Action=1, Next State=31, Reward=-1, Done=False
Step 9: Action=1, Next State=32, Reward=-1, Done=False
Step 10: Action=1, Next State=33, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 8: Action=1, Next State=31, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 8: Action=1, Next State=31, Reward=-1, Done=False
Step 9: Action=1, Next State=32, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 8: Action=1, Next State=31, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 6: Action=1, Next State=29, Reward=-1, Done=False
Step 7: Action=1, Next State=30, Reward=-1, Done=False
Step 8: Action=1, Next State=31, Reward=-1, Done=False
Step 9: Action=1, Next State=32, Reward=-1, Done=False
Step 10: Action=1, Next State=33, Reward=-1, Done=False
Step 11: Action=1, Next State=34, Reward=-1, Done=False
Step 12: Action=1, Next State=35, Reward=-1, Done=False
Step 13: Action=2, Next State=47, Reward=-1, Done=True
total reward:-13
(venv)
kermi@Kerim MINGW64 /e/Ds699/Hw2
$
```

II Implementation of Temporal Difference (TD) Control (60%)

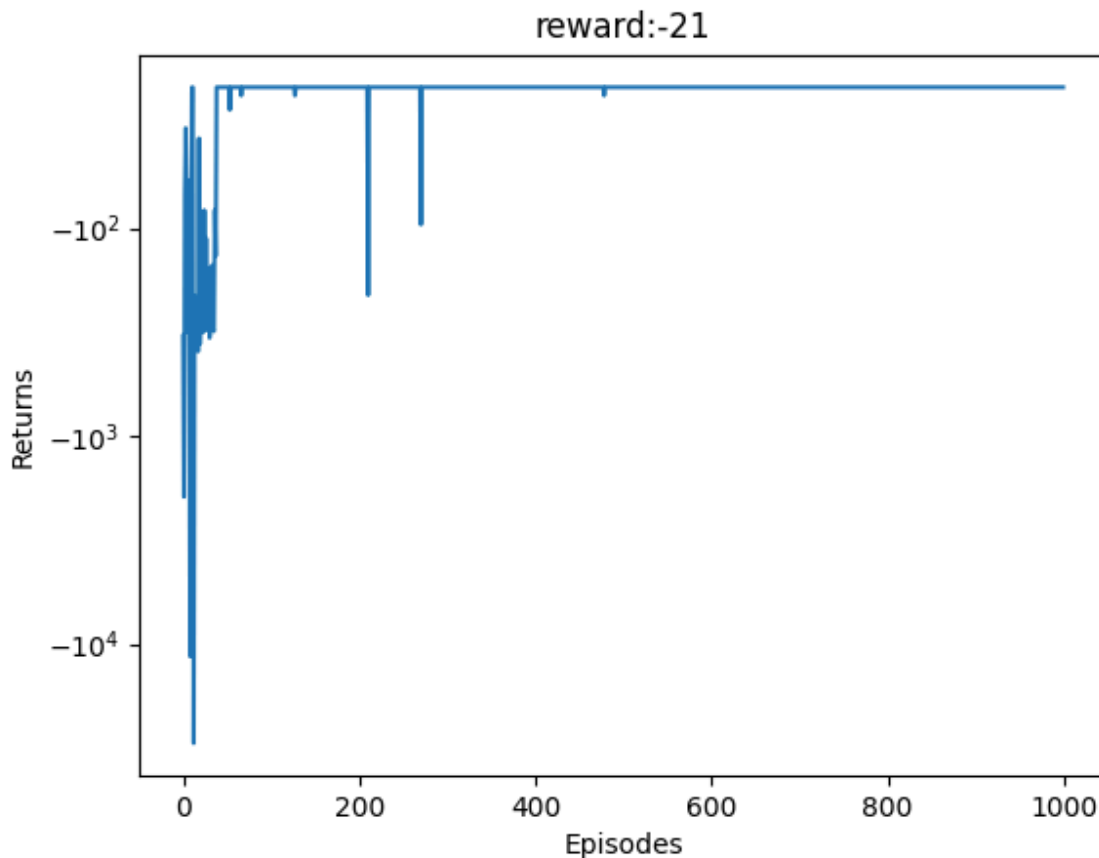
In this section, you need to read through and understand the `codebase_main.py`, `util.py`, `get_args.py` and `codebase_Sarsa.py` in the project. Finish your code in `codebase_Sarsa.py` between the pound sign lines denoted with "Your Code" in each function. Run the program in `codebase_main.py` to test your code and obtain your results.

Please read the comments of each variable carefully and understand the program structure, the definition of inputs and outputs, and the notes under "Your Code".

(a) Implementation of n-step SARSA

- Before implementing the function of `sarsa` and `n_step` in `codebase_Sarsa.py`, you need to finish `epsilon_greedy` function in `codebase_train.py`. Please fully understand the epsilon greedy algorithm and follow the guide under "Your Code".
 - Then please call `epsilon_greedy` in `n_step` function located in the `codebase_Sarsa.py` to finish the `n` steps-sampling to calculate the reward. You need to call `env.step` to finish the `n_step` function. Moreover, you need to take care of the case that the environment terminates before the `n`-step. In this case, you need to stop and record the actual number of steps that have been taken in the variable `acted_steps`. The `acted_steps` is needed in the `sarsa` function for updating the q-table. Please check the definition of input and output carefully under the function names.
 - Then call `n_step` function to finish n-step SARSA in `sarsa` function. Please follow the guide under "Your Code".
 - Set the parameter `method` in `get_args.py` as `sarsa`. Run the program in `codebase_main.py`, and the code will call the function `plot_return` to save a plot of how the total reward changes with the increase of the number of episodes.
 - You can use the `evaluate_policy` in `codebase_train.py` to evaluate your q-table. It can use the greedy method on the q-table to generate a policy, reset the environment, and run the policy on the environment to get an episode. It can print out the policy for the states and the total reward of the episode. The codebase calls `evaluate_policy` function by default.
-
- The code already defined a variable `tabular_q` as a table to store the q values. Each `tabular_q[state][action]` stores the q value for each pair of state and action.
 - In this question, the epsilon is fixed as the value of `init_epsilon=0.1`.

(1) Please paste the generated plot of your result.



(2) Please take a screenshot of the policy and the episode reward after calling the `evaluate_policy`.

```
actions:
[1, 2, 3, 3, 3, 3, 1, 3, 1, 1, 1, 2]
[0, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 2]
[0, 1, 1, 0, 3, 0, 0, 0, 3, 2, 2, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -21.000000
mean and std: (-21, 0)
(verbose)
```

(3) **Question:** According to the generated plot, does the q-table converge? Why if it does not?

According to the generated plot, the Q-table **does converge**. The total episode rewards stabilize around -15 after approximately 150 episodes, indicating that the agent has learned a consistent policy. Although there are occasional dips in reward due to exploration or bad steps, the overall trend shows that learning has plateaued, which is a clear sign of convergence.

(4) **Question:** What are the disadvantages of using a constant value of ϵ ?

If ϵ is a constant value the disadvantages could be a high exploration or low exploration. The chances of getting a perfect ϵ value as a constant is very unlikely and would lead to an either

high or low exploration. If the exploration is high the agent will continue to explore even after it has found a good policy. If the exploration is low the agent will stop before it has found a good policy.

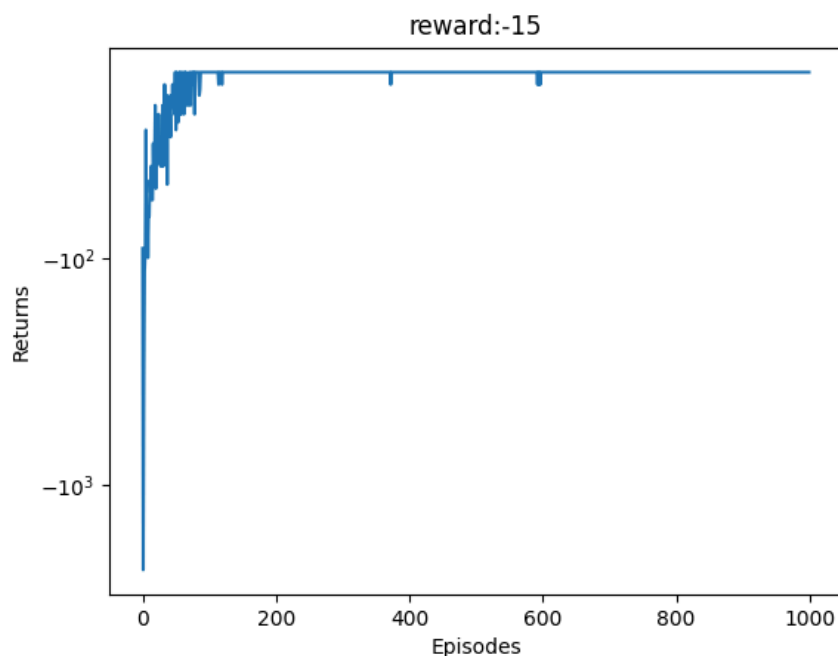
(5) **Question:** Should the value of ϵ gradually increase or decrease as the number of episodes increases?

ϵ should decrease over time because in the early stages we want the agent to explore more and towards the end stages explore less. If it was the opposite it would take longer to complete because the exploration would happen too late and would probably require more episodes.

(b) Optimization of epsilon ϵ

- In this question, we adjust the value of ϵ to optimize the SARSA model and make it converge. According to the convergence condition, please use $\epsilon = \frac{\text{init_epsilon}}{k}$ to improve the algorithm. The `init_epsilon` is the default setting of epsilon in the `get_args.py`, and k is the k th episode generated in the `sarsa` function. Please implement this method in the `sarsa` function in `codebase_Sarsa.py` by replacing the line "`epsilon = init_epsilon`" with your code.
- Please modify the name of the saved plot accordingly if you do not want to overwrite the previously generated plot.

(1) Please paste the generated plot of your result.



(2) Please take a screenshot of the policy and the episode reward.

```

actions:
[1, 1, 1, 1, 1, 1, 3, 1, 1, 2, 2, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2]
[1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -15.000000
mean and std: (-15, 0)

```

(3) **Question:** Please explain why this method can help SARSA to converge.

Promotes early exploration and late convergence since we are dividing by initial epsilon by the number of episodes. This is because it helps SARSA find a stable policy by reducing the randomness.

(4) **Question:** Is the total reward generated from part II.(b) the highest q value of this game? In other words, does SARSA generate the optimal policy with this setting?

Yes the reward generated is better because it finds the optimal policy in fewer steps. It accomplishes this in 15 steps compared to 21 steps.

(c) Effect of number of steps in n-step SARSA

- A computer science student Steve conducted an experiment setting the value of `num_steps` to be 10 and 100, and respectively obtained the reward results and policy results shown in screenshots of Figures 1 and 2 . Based on his results, please finish the following questions.

(1) **Question:** With the state-action pair (1,2) and (17,1), what will happen?
(Hint: state-action pair is defined as (state number, action number). For example, the state-action pair (1,2) indicates that the agent's action is 2 at state 1, located at [0,1].)

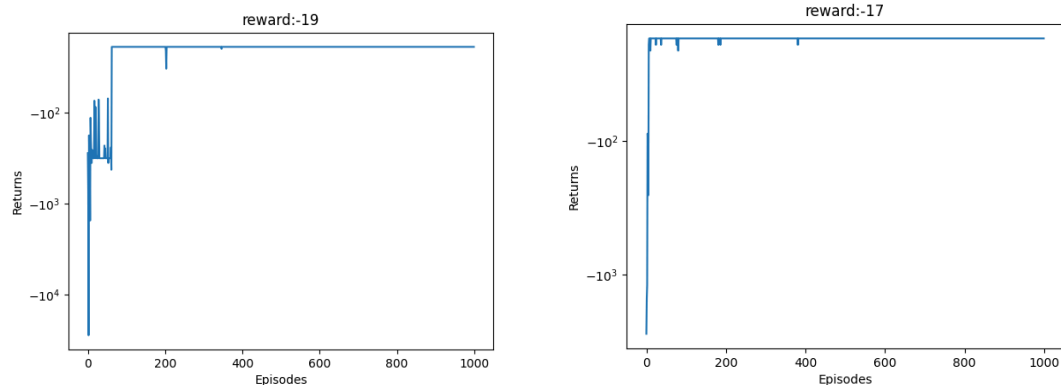
```

--- Inspecting Q-values for Question 1 ---
Q-values for state 1: [-7.201969 -7.13310154 -7.13595549 -7.27219804]
Q-values for state 17: [-5.94308945 -5.6953279 -5.84305638 -6.00667434]
❖ (venv)

```

- At state 1, taking action 2 (Down) leads the agent toward the cliff area. It has a slightly lower Q-value than the best option (Right), which means it's a bad move.
- At state 17, action 1 (Right) is the optimal move, with the highest Q-value among the four options.

(2) **Question:** How do the different numbers of steps affect the convergence of the q-table? Does a larger number of steps lead to faster or slower convergence?



100 steps take a lot longer to converge because it relies more on the longer sequences of rewards. When I tried to run this it took a lot longer to run the results. The images above show my results of 10 and 50. 10 took less than 10 seconds to complete but 50 steps took around a minute to run. We can also see that the reward is better with the 50 steps but it took a lot longer since it goes through more rewards in the early stages. This is because the agent is taking the faster options since it is not scoping out as much as the 50 step version.

(3) **Question:** Are the total rewards of the 10-step and 100-step SARSA higher or lower than those of the 1-step SARSA?

The 10 step Sarsa has better results because it does not look too far out but also not too short sighted. The 1 step sarsa gets a lower reward but is the fastest, but is not the best because the agent again is looking 1 step at a time causing the model to be more risky. The 100 step takes really long and has similar results to the 10 step in terms of rewards but because of its compute time and unstable learning it also isn't the best choice. The total rewards for the 10 and 100 step are higher than those of the 1 step Sarsa.

(4) **Question:** There is an "wrong" action in the policy of the 10-step SARSA shown in the screenshot of Figure 1. Can you find the state-action pair and explain why the policy finally selected the wrong action for this state? You can print out and analyze the q values of all the 4 actions of this state.

(Hint: "wrong" means that the action will lead the agent to get a worse reward.)

From the image we can see that action 2 on the first row looks out of place. The first row is primarily 1 (Right) and we can see that there are two 2's (Down) which seems out of place. We cannot test if the tabular_q in that location is right or not because we cannot see the exact results in the example but just from the agent actions seems off. We can also see that in 100 it follows the same direction but starts going diagonally down later on at step 5. From my code example this is the results I get where the best action is right (6.56) but the action chosen in the example is down (6.73)

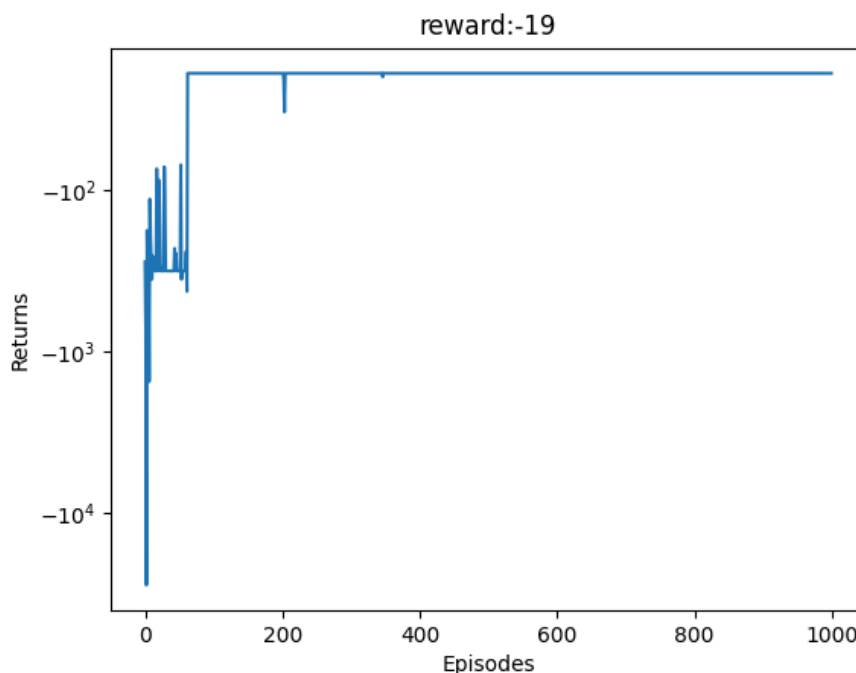
```
--- Q-values for Question ---
Q-values for state 3: [-6.59395273 -6.56343452 -6.72729804 -6.71127084]
❖ (very)
```

- In this sub-question, we try to find out in which case the SARSA fails to find a policy. To prevent an infinite loop, modify the condition of your while loop to limit the episodes to a maximum of 10000 steps. This means the q table stops updating when the episode is done or the total steps of the episode are greater than 10000. Then set the `num_steps` as 10 and rerun the code.

(5) Please paste the plot. Please take a screenshot of the policy and the total reward.

```
kermi@Kerim MINGW64 /e/Ds699/Hw2
$ python codebase_main.py -method sarsa -num_steps 10

actions:
[1, 1, 2, 2, 3, 3, 1, 1, 2, 1, 2, 3]
[0, 0, 1, 1, 1, 1, 0, 1, 1, 2, 2, 0]
[0, 3, 2, 2, 0, 2, 0, 3, 3, 1, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -19.000000
mean and std: (-19, 0)
(venv)
```



- (6) **Question:** Are there any traps in the policy based on the screenshot you got in II.(c)(5)? If so, could you provide the state-action pairs associated with these traps? (Hint: A trap means the policy might lead the environment to a dead end, resulting in non-termination.)

I believe I found 2 traps in this experiment.

- State 4 the agent goes left bringing the agent to stage 3 but then at stage 3 the agent goes down to state 5 and then the agent goes left going back to state 4 causing a circle.
- Another example is state 27 the agent goes down to state 39 but then goes back up to state 27 causing a back and forth.

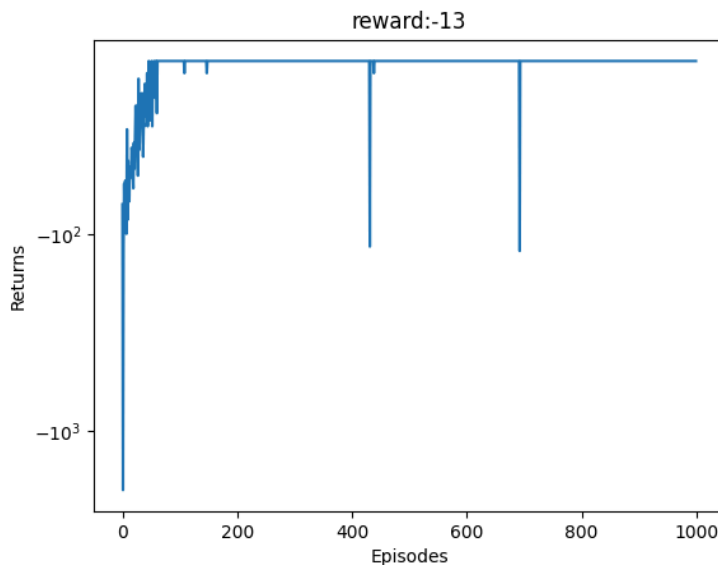
III Implementation of Q-learning (30%)

In this section, you need to read through `main.py`, `util.py`, `get_args.py`, and `codebase_Q_learning.py`. Finish your code in the function `q_learning` in `codebase_Q_learning.py` between the pound sign lines denoted with "Your Code" in each function. Run the program in `codebase_main.py` to test your code.

(a) Implementation of Q-learning

- Please reserve your optimization of ϵ in and by copying your implementation of I.(b) to the same place in the function of `q_learning`. Reset the `init_q_value` as 0.1 and seeds as 1. Set the `method` as `q-learning`
- To implement Q-learning, you need to call `epsilon_greedy` function in `codebase_train.py` (n-step is not needed in this question).
- The implementation is quite similar to SARSA. You just need some simple modifications. Please follow the guide under "Your Code".

(1) Please paste the generated plot. Please take a screenshot of the policy and the total reward.



```
actions:
[2, 0, 1, 1, 1, 3, 2, 1, 0, 2, 3, 2]
[2, 1, 1, 0, 1, 1, 1, 3, 2, 1, 1, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -13.000000
mean and std: (-13, 0)
(venv)
kermi@Kerim MINGW64 /e/Ds699/Hw2
```

- (2) **Question:** Compared with the result you got from partIII.(b)(1), does Q-learning converge faster or slower? Provide your explanation for why it is faster or slower. (Hint: faster or slower here refers to `num_step`, not the actual running time.)

Q-learning is faster because as we can see in the graph it converges at a faster point (around 150) and then there are a few dips after but is relatively settled where Sarsa floats up and down until the 200 point and then settles after.

(3) **Question:** Does the total reward have a higher total reward than the SARSA? Please explain why it is higher or lower.

Q-learning has a higher reward of -13 then Sarsa at -15. Since q-learning is more aggressive at finding the optimal actions it is faster at finding the convergence and settling faster. Sarsa is more cautious and explores more, making it take longer to settle.

(4) **Question:** Is the total reward generated here the highest of the cliff walking?

Yes -13 is the highest generated reward because it avoids the cliff entirely and makes no mistake.

(b) Effect of Alpha α

- Continuing with Steve's experiment, he tested for different values for α to see the effect on Q-learning. He set α to be 0.1 and 0.9, and obtained the reward results separately, as shown in Figure 3. Based on his results, please finish the following question.

(1) **Question:** Do the different values of α affect the total reward of Q-learning? Please explain the reason.

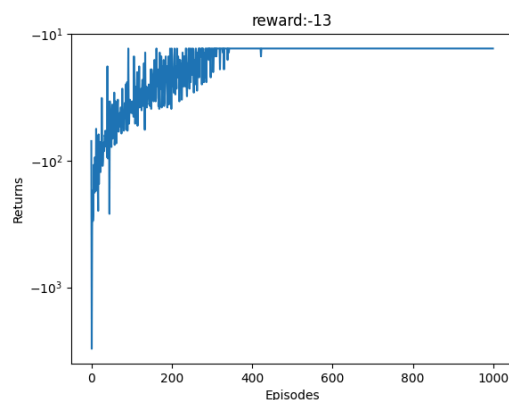
0.1

```
actions:
[2, 0, 0, 1, 3, 1, 3, 0, 1, 1, 1, 0]
[1, 1, 1, 0, 1, 1, 2, 1, 3, 2, 2, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -13.000000
mean and std: (-13, 0)
(venv)
```

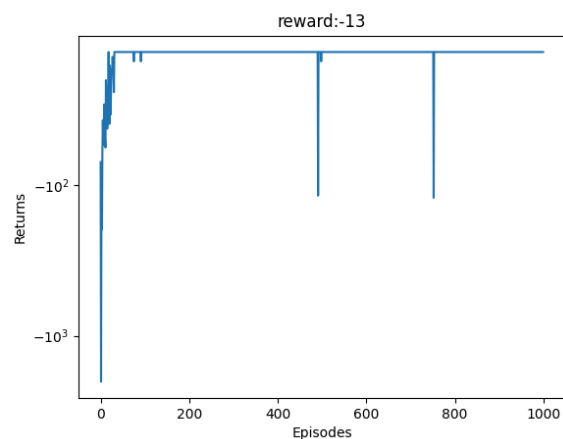
0.9

```
actions:
[1, 1, 1, 1, 1, 1, 3, 3, 0, 1, 1, 2]
[3, 0, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -13.000000
mean and std: (-13, 0)
```

0.1



0.9



When alpha is 0.1 q-values updates slowly which makes the model stable but slower. The total reward slowly improves and converges with a reward -13.

For alpha at 0.8 the q-values update a lot faster and allows the agent to reach the reward quicker. It cause more jumps in the graph but reaches the reward a lot faster.

Low alpha is stable but slower and high alpha is faster but less stable.

(2) **Question:** Which α help Q-learning to converge faster? A larger α or a smaller α ? Can you explain why?

Low alpha is stable but slower and high alpha is faster but less stable. High alpha gives more weight and quickly adapts the q-values based on the rewards. It is less stable however because it can overshoot the weight. For a smaller alpha it updates the weight in smaller numbers which makes it take less jumps and is more stable but is ultimately slower.

(3) **Question:** Do you observe fluctuation in the plot after convergence? Can you explain how α affects the stability of the q-table values?

For fluctuation in the plots for a high alpha because it makes larger updates to q-values based on weights and the rewards causing the graph to have those big dips after convergence. For smaller alpha it is smoother and does not have those drops because it is more stable because the weight and rewards have small adjustments making the graphs be more consistent but has a slower convergence.