Untitled1

May 6, 2023

1 MDP assignment

First of all I am running the primary code and visualize the average_cumulative_reward in order to see that what happens in this algorithm.

After that I will add the described things in the description of the assignment and check the effect and the improvement of each change.

[1]:
```
def running_average(nums):
    result = []
    sum_so_far = 0
    for i, num in enumerate(nums):
        sum_so_far += num
        result.append(sum_so_far / (i + 1))
    return result
```

[11]:
```
import random

from ice import *
import matplotlib.pyplot as plt
import numpy as np

def average_cumulative_reward_visualization(average_cumulative_rewards,␣
↪timesteps):
    window_size = 1000
    windowed_rewards = [np.mean(average_cumulative_rewards[i:i+window_size])
                        for i in range(0, len(average_cumulative_rewards),␣
↪window_size)]
    plt.plot(windowed_rewards)
    plt.xlabel('Time')
    plt.ylabel('Average cumulative reward')
    plt.title('Average cumulative reward ' + str(timesteps) + " EPISODES")
    plt.yticks(np.arange(0, max(windowed_rewards), 5))
    plt.rcParams['figure.figsize'] = [20, 20]
    plt.show()
```

[12]:

```
def␣
↪average_of_average_cumulative_reward_visualization(average_cumulative_rewards,␣
↪timesteps):
    x = running_average(average_cumulative_rewards)
    plt.plot(running_average(x))
    plt.xlabel('Time')
    plt.ylabel('Average cumulative reward')
    plt.title('Average of average cumulative reward ' + str(timesteps) + "␣
↪EPISODES until each episod")
    plt.yticks(np.arange(0, max(x), 5))
    plt.rcParams['figure.figsize'] = [20, 20]
    plt.show()
```

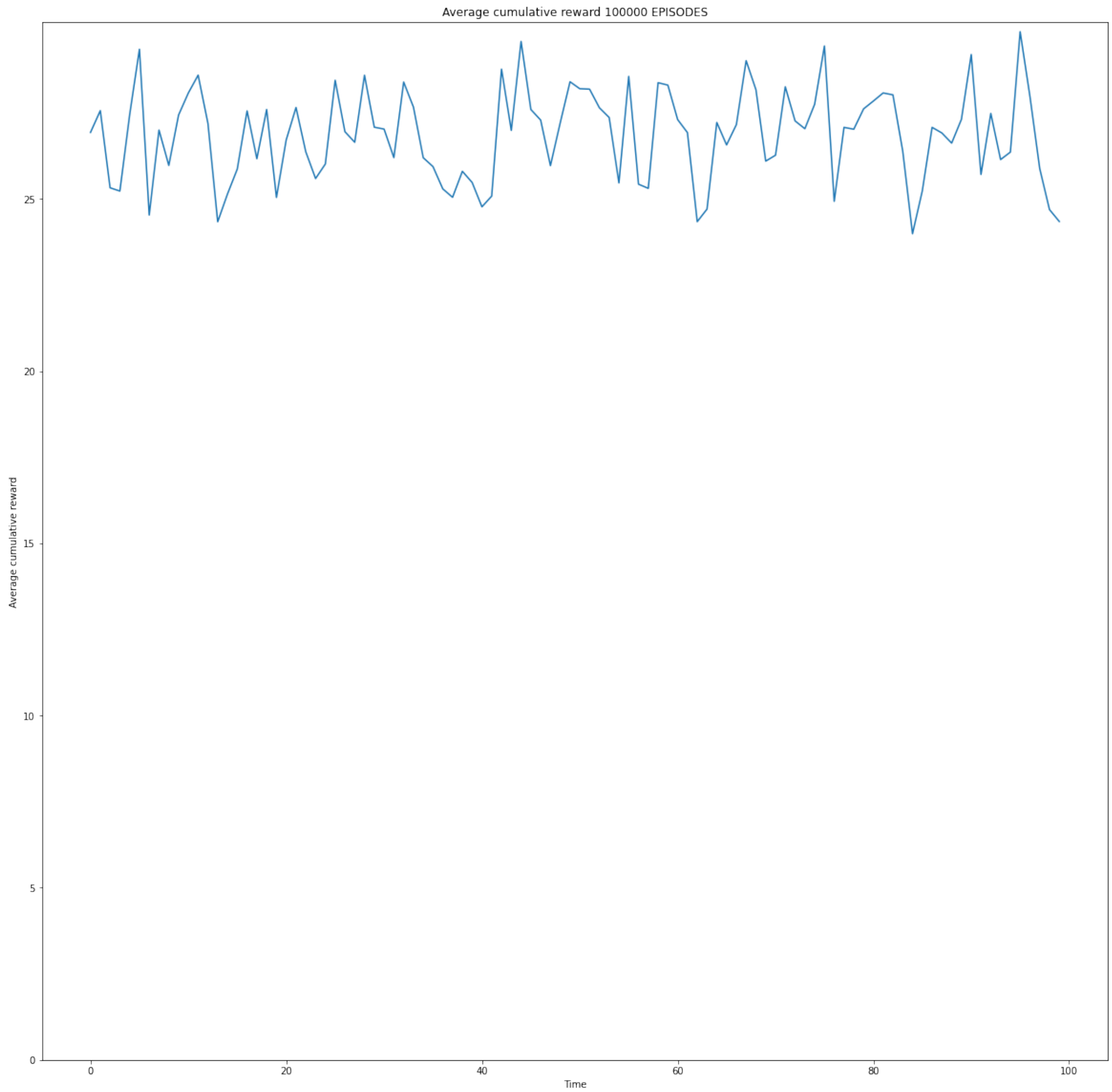
1.1 Primary code

This code is the primary code provided in the assignment. Let see what happens.

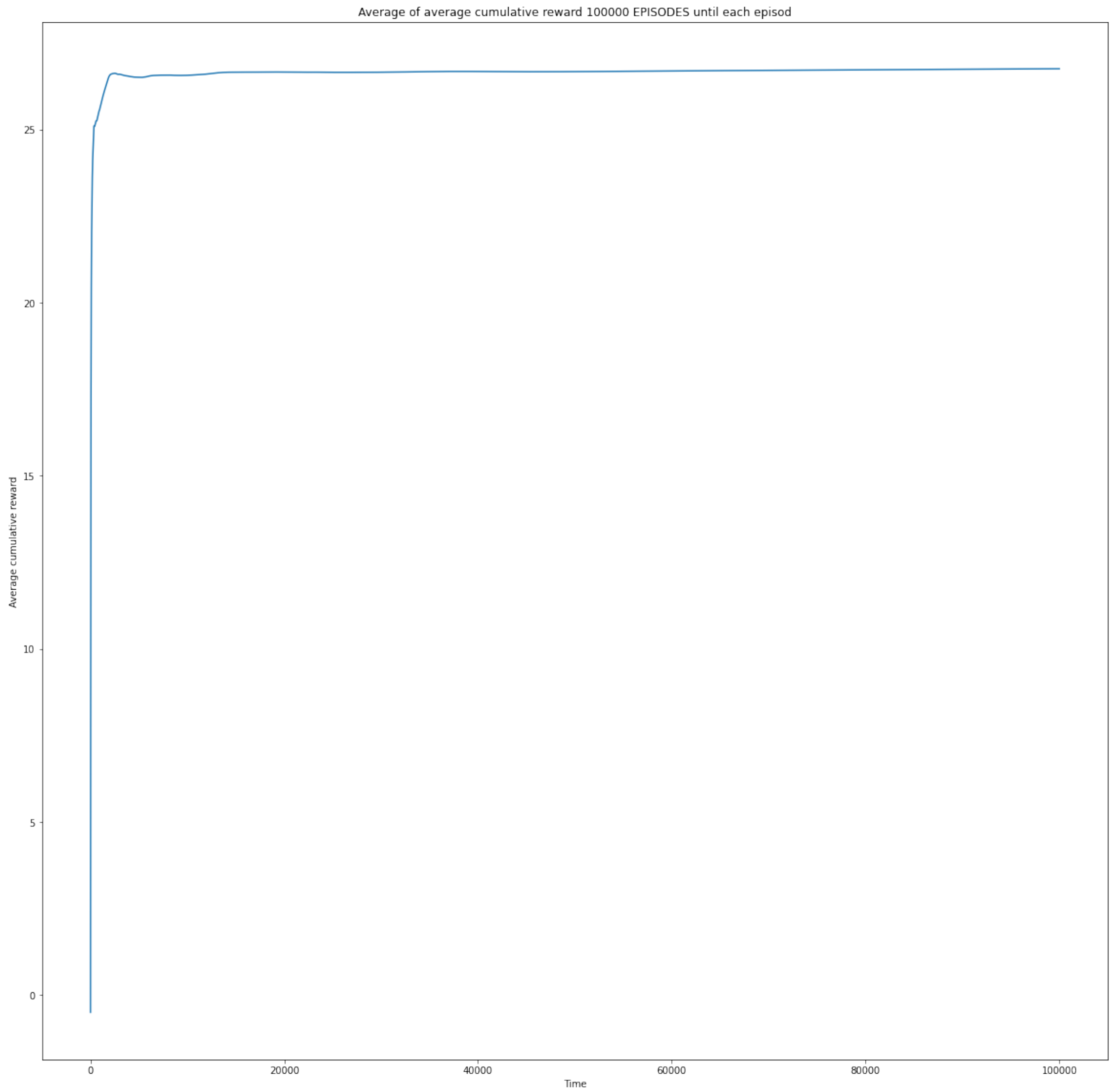
[13]:
```
from ice import *
EPISODES = 100000
EPSILON = 0.1
GAMMA = 0.9
LEARNING_RATE = 0.1
average_cumulative_rewards = []

def argmax(l):
    """ Return the index of the maximum element of a list """
    return max(enumerate(l), key=lambda x:x[1])[0]

def main():
    env = Ice()
    average_cumulative_reward = 0.0
    # Q-table, 4x4 states, 4 actions per state
    qtable = [[0., 0., 0., 0.] for state in range(4*4)]
    # Loop over episodes
    for i in range(EPISODES):
        state = env.reset()
        terminate = False
        cumulative_reward = 0.0
        # Loop over time-steps
        while not terminate:
            # Compute what the greedy action for the current state is
            a = 0
            # Sometimes, the agent takes a random action, to explore the␣
↪environment
            if random.random() < EPSILON:
                a = random.randrange(4)
            # Perform the action
```

```
            next_state, r, terminate = env.step(a)
            # Update the Q-Table
            qtable[state][a] = 0.0
            # Update statistics
            cumulative_reward += r
            state = next_state
        # Per-episode statistics
        average_cumulative_reward *= 0.95
        average_cumulative_reward += 0.05 * cumulative_reward
        average_cumulative_rewards.append(average_cumulative_reward)
        if i%(EPISODES/10) == 0:
            print(i, cumulative_reward, average_cumulative_reward)
    # Print the value table
    for y in range(4):
        for x in range(4):
            print('%03.3f ' % max(qtable[y*4 + x]), end='')
        print()
    average_cumulative_reward_visualization(average_cumulative_rewards, EPISODES)
   ␣
 ↪average_of_average_cumulative_reward_visualization(average_cumulative_rewards,␣
 ↪EPISODES)
if __name__ == '__main__':
    main()
```

```
0 -10.0 -0.5
10000 100.0 25.983389338321487
20000 -10.0 28.208808476567246
30000 100.0 27.157563581070434
40000 -10.0 15.514502419699102
50000 -10.0 24.179136555152255
60000 -10.0 12.942133321097577
70000 100.0 28.858754516945076
80000 100.0 26.71151041105892
90000 100.0 27.931959601383646
0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000
```



Average cumulative reward 100000 EPISODES
Average cumulative reward
Time
0
5
10
15
20
25
0
20
40
60
80
100



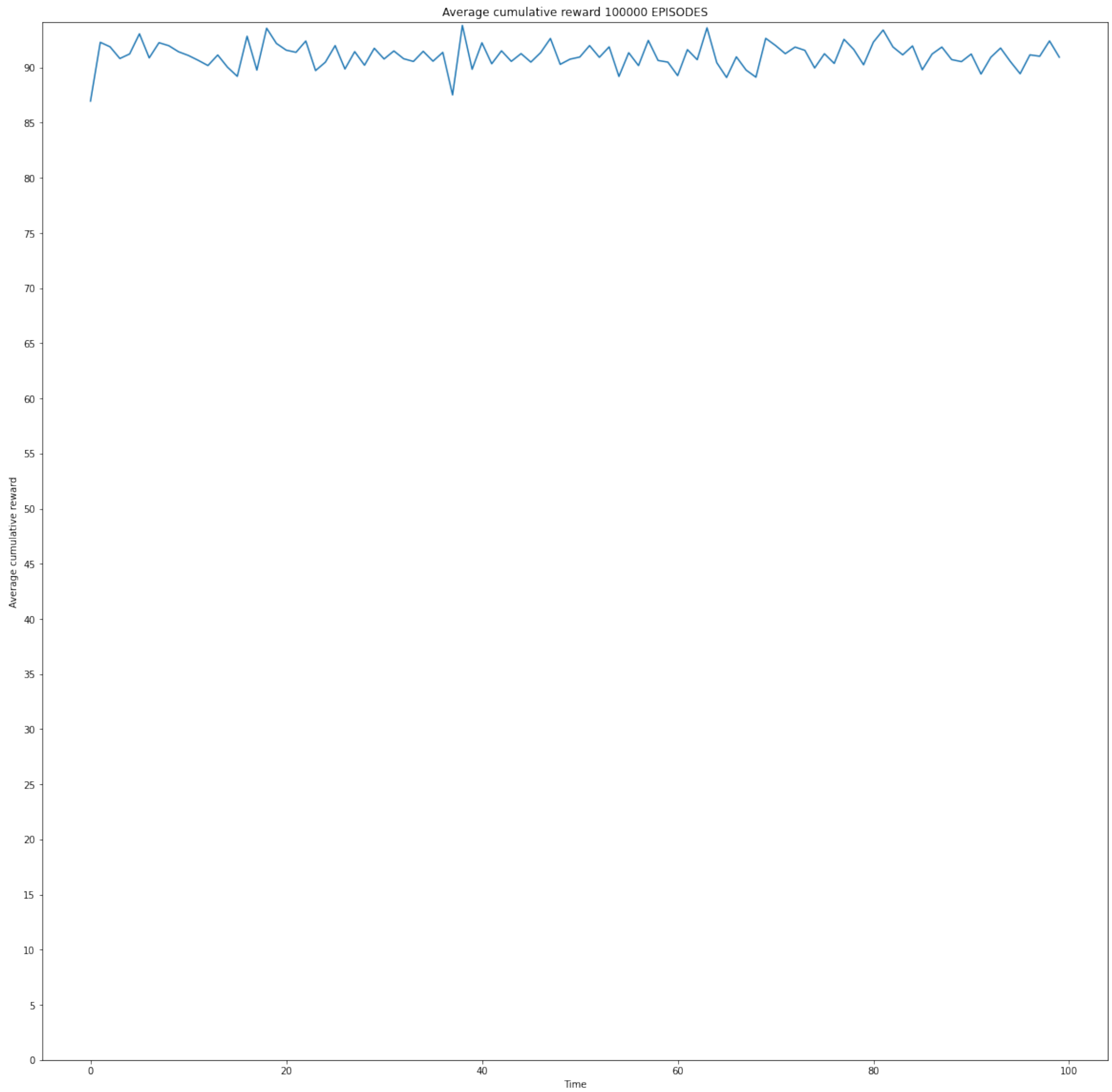
As you can see there is no sign of learning and because there is not any. I also implemented a function for ploting. In this function we average every 1000 point and show them in order to have a more clear plot. ## Add Q learning

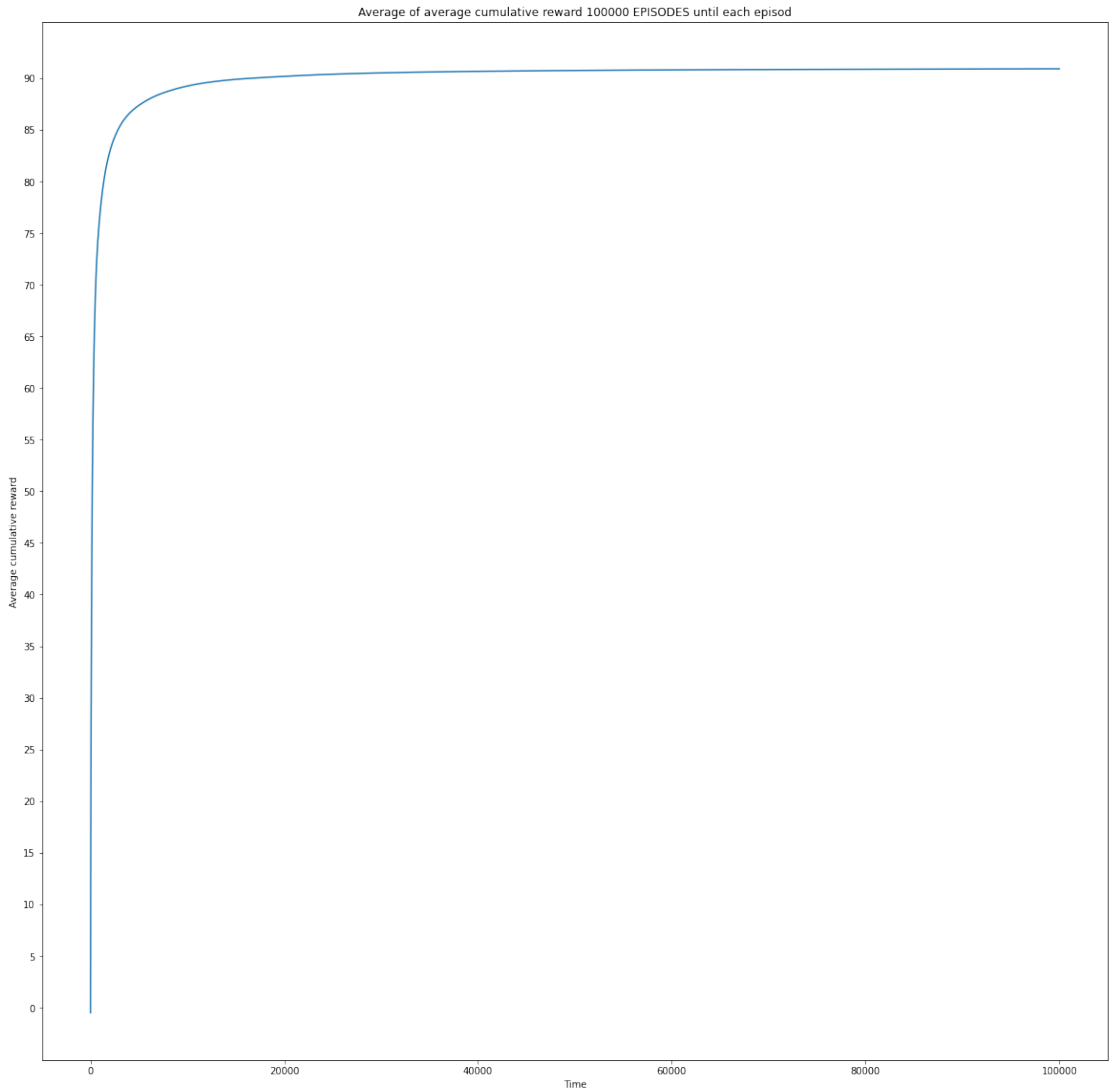
[14]:
```
EPISODES = 100000
EPSILON = 0.1
GAMMA = 0.9
LEARNING_RATE = 0.1
average_cumulative_rewards = []

def argmax(l):
    """ Return the index of the maximum element of a list
    """
    return max(enumerate(l), key=lambda x:x[1])[0]
```

```
def main():
    env = Ice()
    average_cumulative_reward = 0.0

    # Q-table, 4x4 states, 4 actions per state
    qtable = [[0., 0., 0., 0.] for state in range(4*4)]

    # Loop over episodes
    for i in range(EPISODES):
        state = env.reset()
        terminate = False
        cumulative_reward = 0.0

        # Loop over time-steps
        while not terminate:
            # Compute what the greedy action for the current state is
            # UPDATED: We choose the best action in a state based on the q tabla
            a = argmax(qtable[state])

            # Sometimes, the agent takes a random action, to explore the␣
↪environment
            if random.random() < EPSILON:
                a = random.randrange(4)

            # Perform the action
            next_state, r, terminate = env.step(a)

            # Update the Q-Table
            # UPDATED:
            Next_Best_Action = argmax(qtable[next_state])
            qtable[state][a] = qtable[state][a] + LEARNING_RATE * (r + GAMMA *␣
↪qtable[next_state][Next_Best_Action] - qtable[state][a])

            # Update statistics
            cumulative_reward += r
            state = next_state

        # Per-episode statistics
        average_cumulative_reward *= 0.95
        average_cumulative_reward += 0.05 * cumulative_reward
        average_cumulative_rewards.append(average_cumulative_reward)
        if i % (EPISODES / 10) == 0:
            print(i, cumulative_reward, average_cumulative_reward, EPSILON)

    # Print the value table
    for y in range(4):
```

```
        for x in range(4):
            print('%03.3f ' % max(qtable[y*4 + x]), end='')

        print()

    average_cumulative_reward_visualization(average_cumulative_rewards, EPISODES)
    ␣
↪average_of_average_cumulative_reward_visualization(average_cumulative_rewards,␣
↪EPISODES)

if __name__ == '__main__':
    main()
```

```
0 -10.0 -0.5 0.1
10000 100.0 89.09462398854957 0.1
20000 100.0 86.47647842209874 0.1
30000 100.0 85.29678212906015 0.1
40000 100.0 87.25015745823717 0.1
50000 100.0 95.14890420910311 0.1
60000 120.0 92.6066406395792 0.1
70000 120.0 98.31055939129786 0.1
80000 100.0 80.45455686500311 0.1
90000 100.0 94.35302217612622 0.1
83.780 91.284 100.000 0.000
73.507 0.000 90.000 0.000
66.545 60.171 81.668 0.000
60.663 0.000 0.000 0.000
```


Average cumulative reward 100000 EPISODES
Average cumulative reward
Time
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
0
20
40
60
80
100



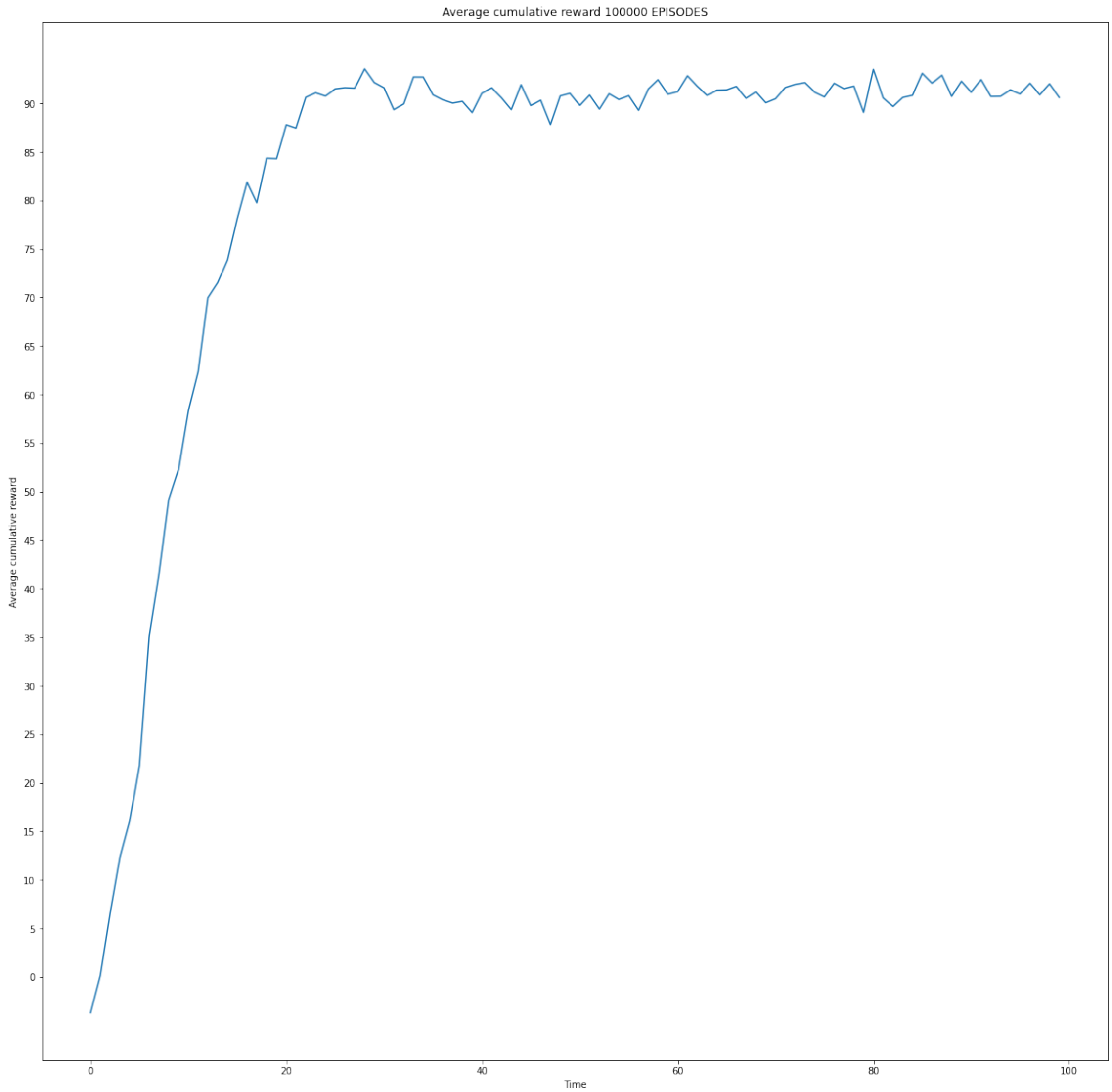
This one is looking better with higher values but because epsilon is very small and does not change the learning is not visible in the plot and also its limited. ## Add Epsilon decay In this part I applied the epsilon decay with DECAY = 0.9999, min_EPSILON = 0.1 and primary EPSILON = 1.

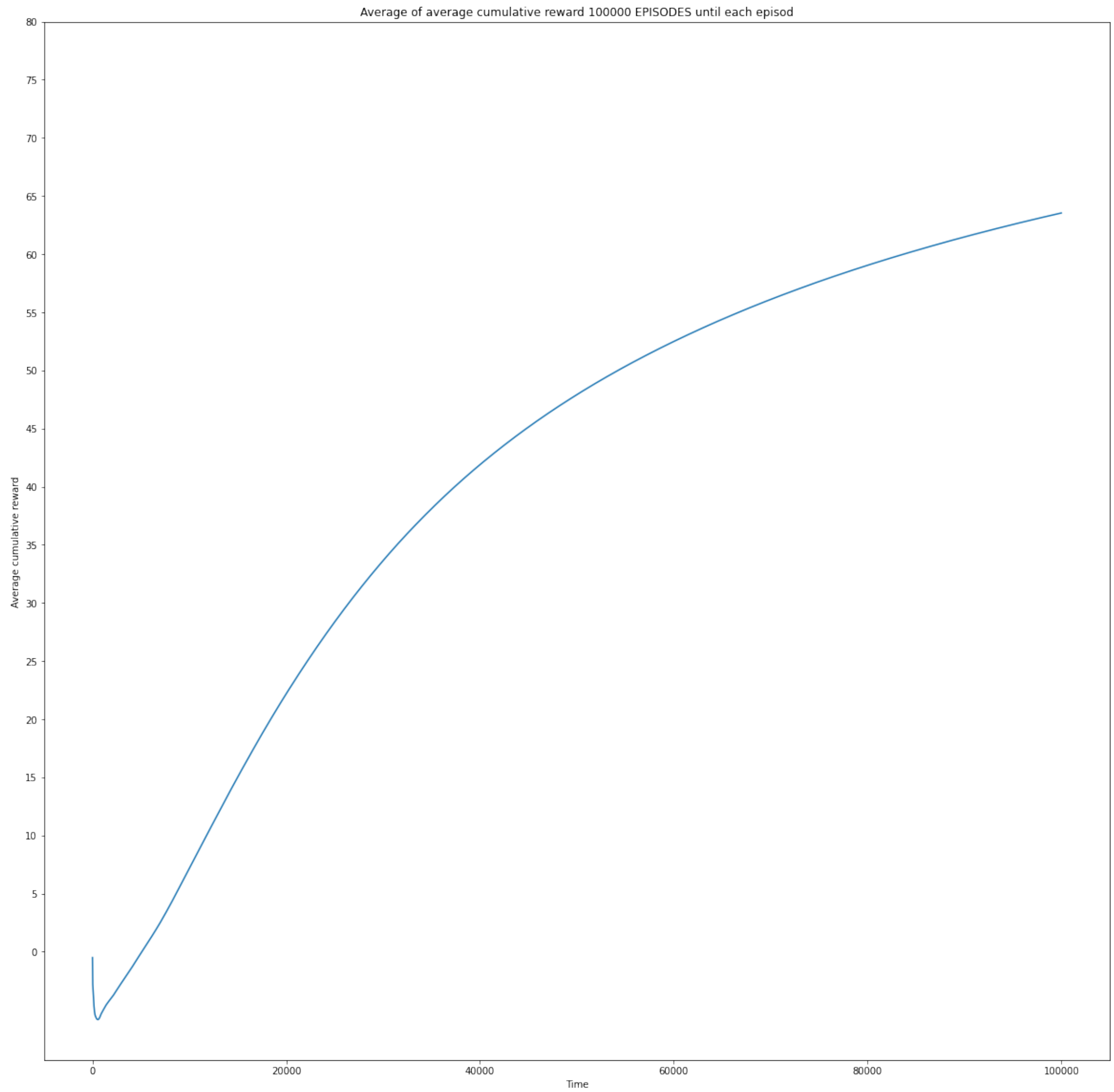
[15]:
```
EPISODES = 100000
EPSILON = 1
min_EPSILON = 0.1
DECAY = 0.9999
GAMMA = 0.9
LEARNING_RATE = 0.1
average_cumulative_rewards = []

def argmax(l):
```

```python
    """ Return the index of the maximum element of a list
    """
    return max(enumerate(l), key=lambda x:x[1])[0]

def main():
    env = Ice()
    average_cumulative_reward = 0.0

    # Q-table, 4x4 states, 4 actions per state
    qtable = [[0., 0., 0., 0.] for state in range(4*4)]

    # Loop over episodes
    for i in range(EPISODES):
        state = env.reset()
        terminate = False
        cumulative_reward = 0.0

        global EPSILON
        EPSILON = max(min_EPSILON, EPSILON*DECAY)

        # Loop over time-steps
        while not terminate:
            # Compute what the greedy action for the current state is
            # UPDATED: We choose the best action in a state based on the q tabla
            a = argmax(qtable[state])

            # Sometimes, the agent takes a random action, to explore the␣
 ↪environment
            if random.random() < EPSILON:
                a = random.randrange(4)

            # Perform the action
            next_state, r, terminate = env.step(a)

            # Update the Q-Table
            # UPDATED:
            Next_Best_Action = argmax(qtable[next_state])
            qtable[state][a] = qtable[state][a] + LEARNING_RATE * (r + GAMMA *␣
 ↪qtable[next_state][Next_Best_Action] - qtable[state][a])

            # Update statistics
            cumulative_reward += r
            state = next_state

        # Per-episode statistics
        average_cumulative_reward *= 0.95
        average_cumulative_reward += 0.05 * cumulative_reward
```

```python
        average_cumulative_rewards.append(average_cumulative_reward)
        if i % (EPISODES / 10) == 0:
            print(i, cumulative_reward, average_cumulative_reward, EPSILON)

    # Print the value table
    for y in range(4):
        for x in range(4):
            print('%03.3f ' % max(qtable[y*4 + x]), end='')

        print()

    average_cumulative_reward_visualization(average_cumulative_rewards, EPISODES)
    ␣
 ↪average_of_average_cumulative_reward_visualization(average_cumulative_rewards,␣
 ↪EPISODES)

if __name__ == '__main__':
    main()
```

```
0 -10.0 -0.5 0.9999
10000 -10.0 62.59279330352855 0.3678242603283259
20000 120.0 84.27321415762967 0.13530821730781062
30000 100.0 92.23425584062925 0.1
40000 100.0 90.93428557135756 0.1
50000 100.0 93.98845692962699 0.1
60000 100.0 87.87326522644378 0.1
70000 100.0 84.6263796954064 0.1
80000 70.0 81.31338057523452 0.1
90000 100.0 90.9661307823188 0.1
81.513 90.192 100.000 0.000
73.832 0.000 90.000 0.000
67.260 91.727 81.002 0.000
61.050 0.000 0.000 0.000
```


Average cumulative reward 100000 EPISODES
Average cumulative reward
Time
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
0
20
40
60
80
100



I also run the program for DECAY = 0.999999 and for 3000000 iteration and have the same results.

[16]:
```
EPISODES = 3000000
EPSILON = 1
min_EPSILON = 0.1
DECAY = 0.999999
GAMMA = 0.9
LEARNING_RATE = 0.1
average_cumulative_rewards = []

def argmax(l):
    """ Return the index of the maximum element of a list
    """
    return max(enumerate(l), key=lambda x:x[1])[0]
```

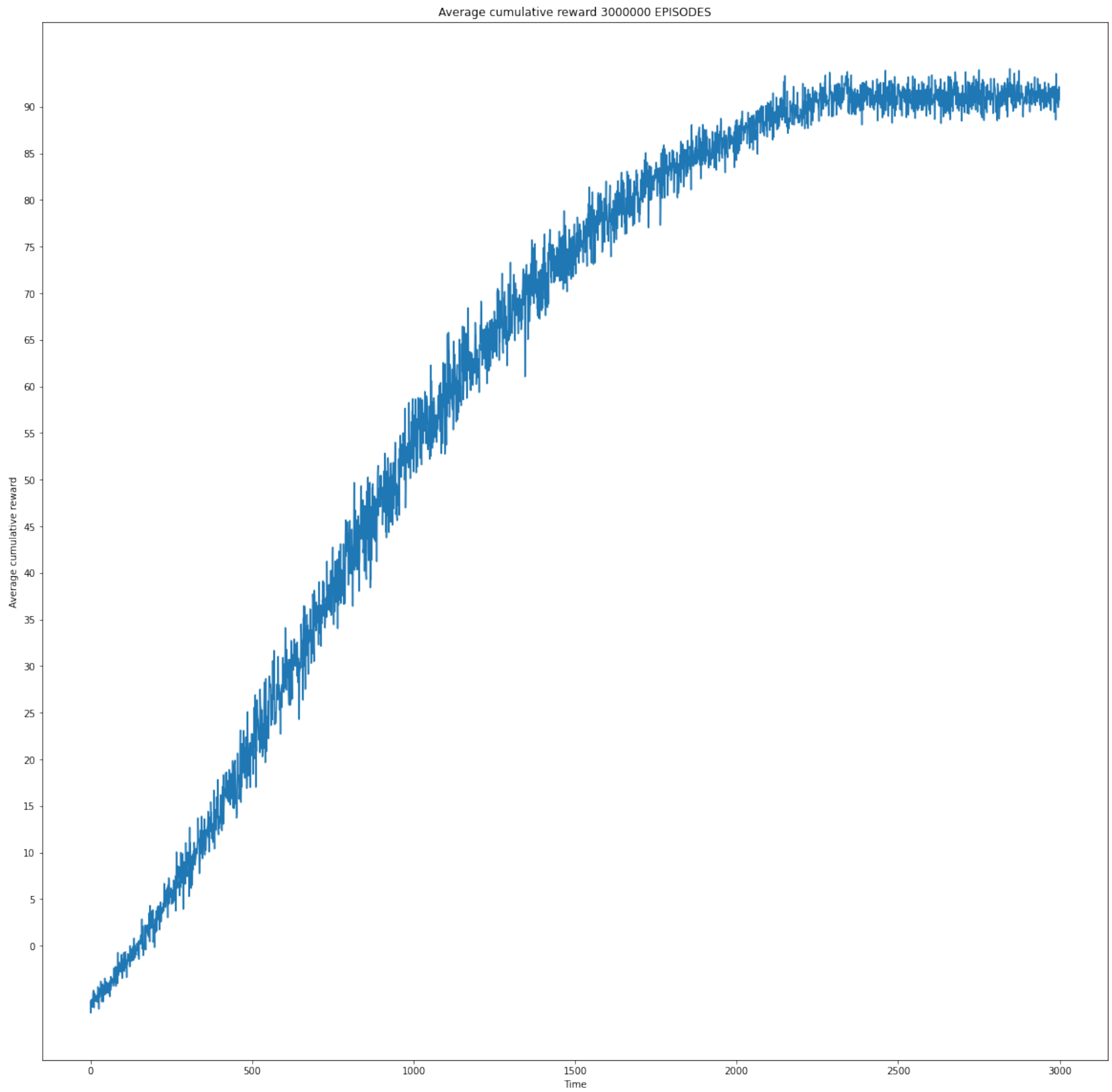
```python
def main():
    env = Ice()
    average_cumulative_reward = 0.0

    # Q-table, 4x4 states, 4 actions per state
    qtable = [[0., 0., 0., 0.] for state in range(4*4)]

    # Loop over episodes
    for i in range(EPISODES):
        state = env.reset()
        terminate = False
        cumulative_reward = 0.0

        global EPSILON
        EPSILON = max(min_EPSILON, EPSILON*DECAY)

        # Loop over time-steps
        while not terminate:
            # Compute what the greedy action for the current state is
            # UPDATED: We choose the best action in a state based on the q tabla
            a = argmax(qtable[state])

            # Sometimes, the agent takes a random action, to explore the␣
 ↪environment
            if random.random() < EPSILON:
                a = random.randrange(4)

            # Perform the action
            next_state, r, terminate = env.step(a)

            # Update the Q-Table
            # UPDATED:
            Next_Best_Action = argmax(qtable[next_state])
            qtable[state][a] = qtable[state][a] + LEARNING_RATE * (r + GAMMA *␣
 ↪qtable[next_state][Next_Best_Action] - qtable[state][a])

            # Update statistics
            cumulative_reward += r
            state = next_state

        # Per-episode statistics
        average_cumulative_reward *= 0.95
        average_cumulative_reward += 0.05 * cumulative_reward
        average_cumulative_rewards.append(average_cumulative_reward)
        if i % (EPISODES / 100) == 0:
            print(i, cumulative_reward, average_cumulative_reward, EPSILON)
```

```python
    # Print the value table
    for y in range(4):
        for x in range(4):
            print('%03.3f ' % max(qtable[y*4 + x]), end='')

        print()

    average_cumulative_reward_visualization(average_cumulative_rewards, EPISODES)
   ␣
 ↪average_of_average_cumulative_reward_visualization(average_cumulative_rewards,␣
 ↪EPISODES)

if __name__ == '__main__':
    main()
```

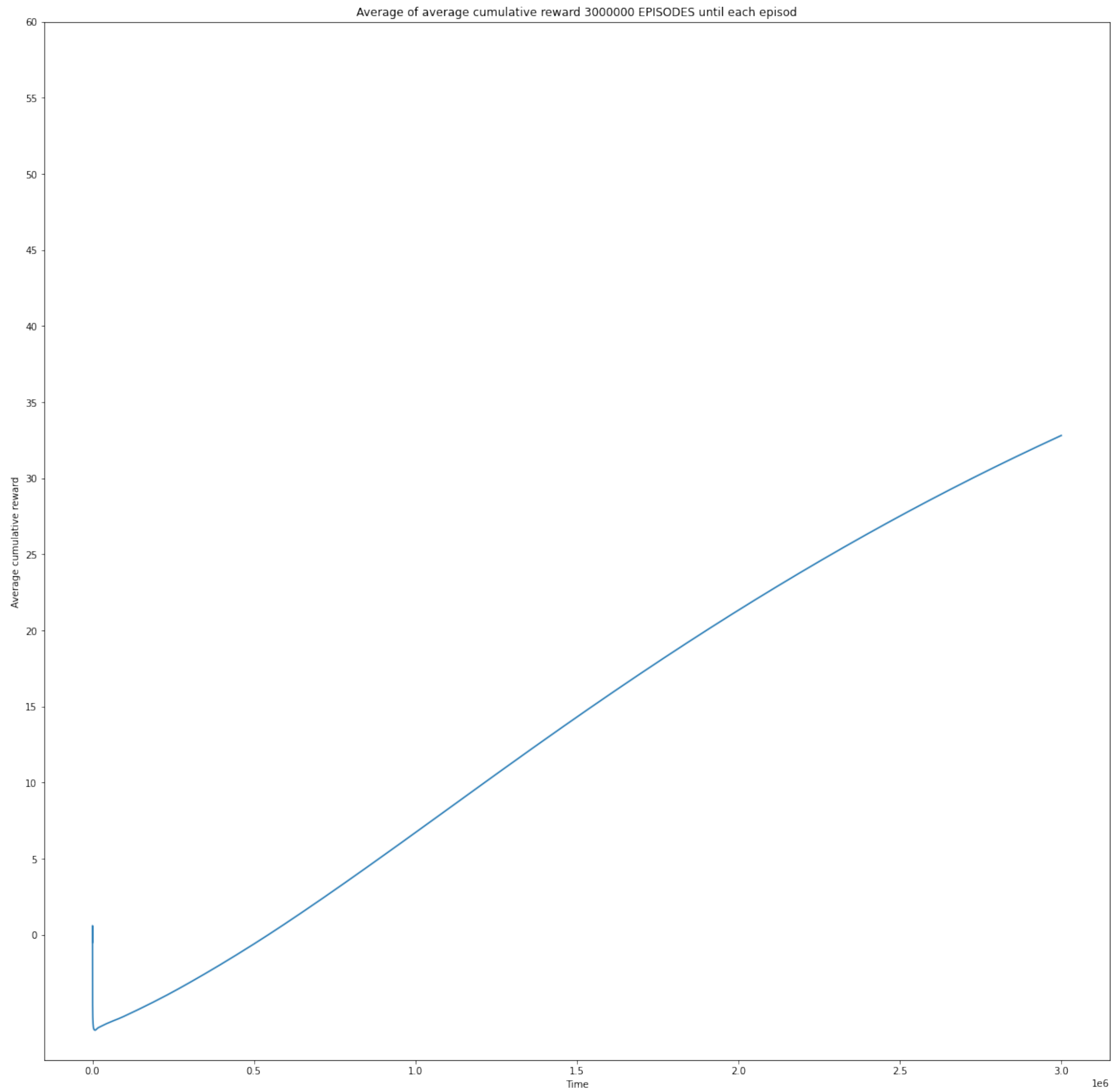
```
0 -10.0 -0.5 0.999999
30000 -10.0 -2.3245609059137826 0.9704445485454593
60000 -10.0 -5.700436830477639 0.941763563565167
90000 -10.0 -4.54907095581898 0.9139302302107998
120000 -10.0 -0.294481467240649 0.8869194965784752
150000 -10.0 6.0428814241566 0.8607070511603203
180000 -10.0 -5.052487140820837 0.8352693009624697
210000 -10.0 7.425383576730465 0.810583350269737
240000 -10.0 4.583608613746494 0.7866269800379648
270000 -10.0 -2.504724152562729 0.7633786278952455
300000 -10.0 2.2431297824007856 0.7408173687344279
330000 10.0 14.527637854228177 0.7189228958790044
360000 -10.0 17.085658739398827 0.6976755028057985
390000 -10.0 12.511336161470522 0.6770560654076723
420000 -10.0 19.892000174071676 0.6570460247805289
450000 100.0 21.085311799268602 0.6376273705190901
480000 -10.0 7.74761698998633 0.6187826245062464
510000 -10.0 21.687170133178597 0.6004948251815632
540000 100.0 34.544047605719186 0.582747512274721
570000 -10.0 21.892908184079594 0.5655247119901355
600000 100.0 15.627552269112645 0.548810922629492
630000 100.0 28.240686556560284 0.5325911006390989
660000 -10.0 26.15881062614805 0.5168506470696865
690000 -10.0 33.81124218903529 0.5015753944363639
720000 10.0 39.12252346108553 0.48675159396690026
750000 120.0 43.703899609507516 0.47236590322689354
780000 -10.0 44.73165763708578 0.4584053741106672
810000 100.0 56.56245691304051 0.44485744118708204
840000 -10.0 22.65897149141996 0.4317099103897973
870000 -10.0 47.536036629102824 0.4189509480417789
900000 100.0 53.933485220901595 0.4065690702041693
930000 120.0 47.43824296855127 0.3945531323399635
```

960000 120.0 50.29124731433442 0.3828923192831714
990000 100.0 61.50146561574738 0.3715761355044158
1020000 10.0 62.73094836580161 0.3605943956642465
1050000 120.0 56.18261902419613 0.3499372154456271
1080000 100.0 64.81802623849904 0.33959500265738624
1110000 120.0 83.02643146668204 0.3295584486005903
1140000 100.0 66.06277751508793 0.3198185196900615З
1170000 30.0 69.73126599983382 0.31036644932354424
1200000 100.0 71.86226404583734 0.3011937299911748
1230000 120.0 85.98562353547206 0.2922921056181172
1260000 100.0 66.37064877377932 0.28365356413353937
1290000 100.0 60.47354394078816 0.27527033025920883
1320000 100.0 59.70864867745859 0.2671348585112179
1350000 100.0 84.75703394174434 0.25923982640850174
1380000 -10.0 77.78458297379895 0.2515781278821273
1410000 -10.0 63.89412233292457 0.24414286687935027
1440000 120.0 67.59788282921654 0.2369273511566766
1470000 10.0 71.5230472312555 0.22992508625640118
1500000 -10.0 61.71787884403483 0.22312976966113832
1530000 100.0 64.65465500276305 0.21653528512114145
1560000 -10.0 76.75866897324222 0.21013569714924638
1590000 120.0 81.72167862852339 0.2039252456785305
1620000 100.0 79.8122536958371 0.19789834087786284
1650000 100.0 79.15991220701251 0.19204955812066862
1680000 100.0 63.94278329427316 0.1863736331023999
1710000 -10.0 73.44850050618456 0.18086545710229307
1740000 100.0 91.14732872370055 0.1755200723851776
1770000 -10.0 84.67340733633047 0.17033266773916778
1800000 120.0 97.4112764185742 0.16529857414525437
1830000 120.0 77.32186719655675 0.1604132605748591
1860000 100.0 88.43822162479034 0.15567232991160668
1890000 100.0 88.24458499595409 0.1510715149936045
1920000 -10.0 65.31757098370669 0.14660667477272363
1950000 100.0 87.30888447570196 0.14227379058735773
1980000 100.0 92.22046612714475 0.13806896254536252
2010000 100.0 92.47806898227229 0.13398840601388148
2040000 100.0 94.90979474774566 0.1300284482129151
2070000 100.0 89.55633882247359 0.12618552490957494
2100000 120.0 75.39382860003438 0.12245617721002164
2130000 100.0 86.52180942124676 0.1188370484462311
2160000 100.0 90.62574587596036 0.11532488115475811
2190000 100.0 78.23390648433929 0.11191651414480247
2220000 120.0 93.54143394412516 0.10860887965291346
2250000 100.0 95.5764016863563 0.10539900058180081
2280000 100.0 94.31309346389364 0.10228398782073586
2310000 -10.0 75.0837415291787 0.1
2340000 100.0 95.5391628593875 0.1
2370000 100.0 93.81361864106697 0.1

```
2400000 100.0 86.08345213801792 0.1
2430000 100.0 91.52378006001494 0.1
2460000 100.0 90.14332835107633 0.1
2490000 100.0 97.28913676746193 0.1
2520000 100.0 90.12169267740478 0.1
2550000 100.0 95.13941535369513 0.1
2580000 100.0 99.41628944285279 0.1
2610000 100.0 98.24253648613814 0.1
2640000 100.0 92.0554155824889 0.1
2670000 100.0 89.20180749889862 0.1
2700000 100.0 86.98554485820902 0.1
2730000 120.0 85.4626687599401 0.1
2760000 100.0 83.93146330681421 0.1
2790000 100.0 89.42368571399125 0.1
2820000 100.0 89.61721888538021 0.1
2850000 100.0 89.18485160527904 0.1
2880000 100.0 90.69982363348909 0.1
2910000 120.0 102.35233942302804 0.1
2940000 100.0 82.59624735775778 0.1
2970000 100.0 93.50194813972631 0.1
81.916 90.028 100.000 0.000
73.838 0.000 90.000 0.000
66.752 71.738 81.316 0.000
60.943 0.000 0.000 0.000
```



Average cumulative reward 3000000 EPISODES
Average cumulative reward
Time
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
0
500
1000
1500
2000
2500
3000


Average of average cumulative reward 3000000 EPISODES until each episod
Average cumulative reward
Time
1e6
0.0
0.5
1.0
1.5
2.0
2.5
3.0
60
55
50
45
40
35
30
25
20
15
10
5
0

[ ]: