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]:

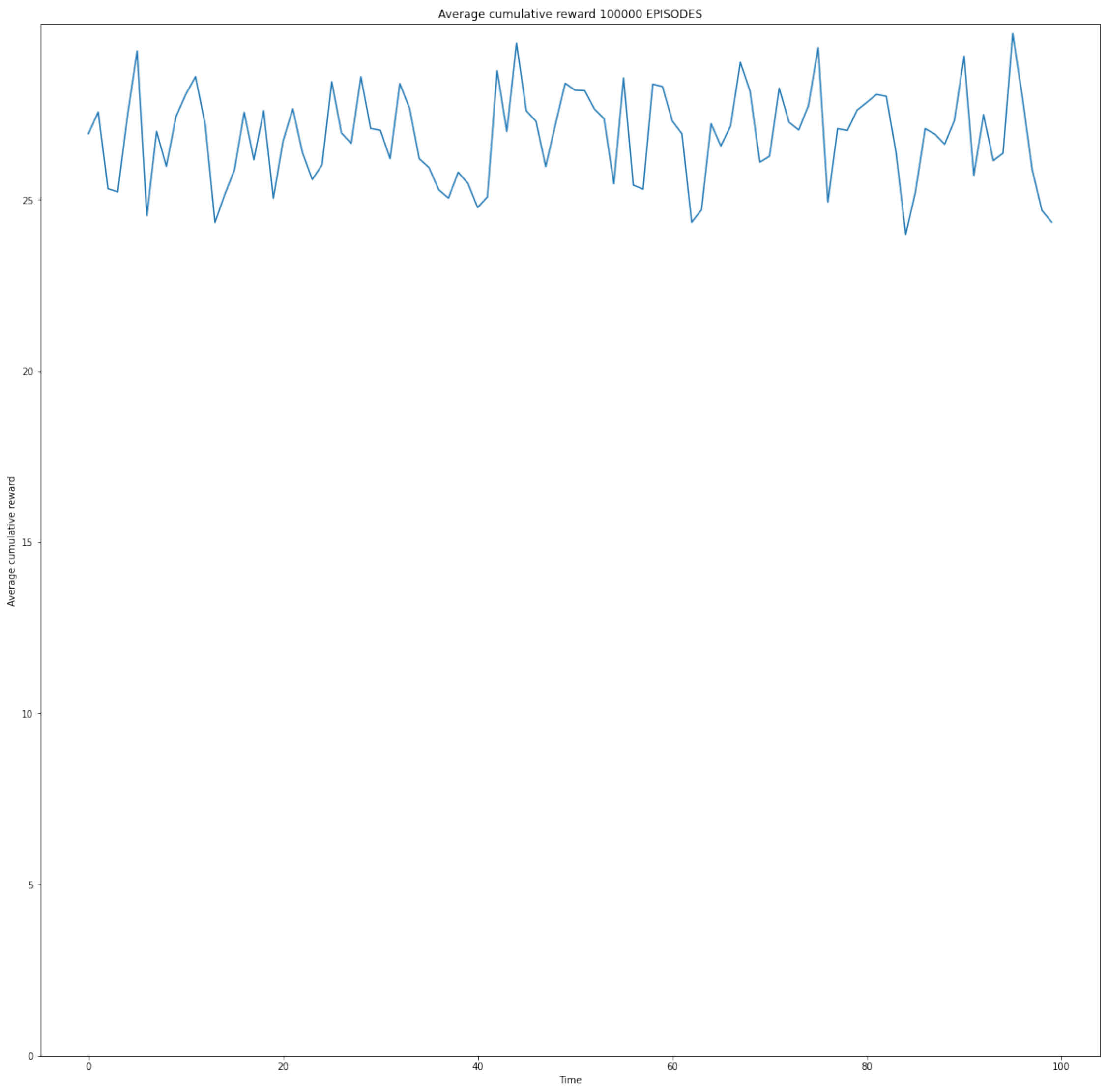
```
def␣
↪average_of_average_cumulative_reward_visualization(average_cumulative_rewards,␣
↪timesteps):
    x = running_average(average_cumulative_rewards)
    plt.plot(running_average(x))
    plt.xlabel('Time')
    plt.ylabel('Average cumulative reward')
    plt.title('Average of average cumulative reward ' + 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

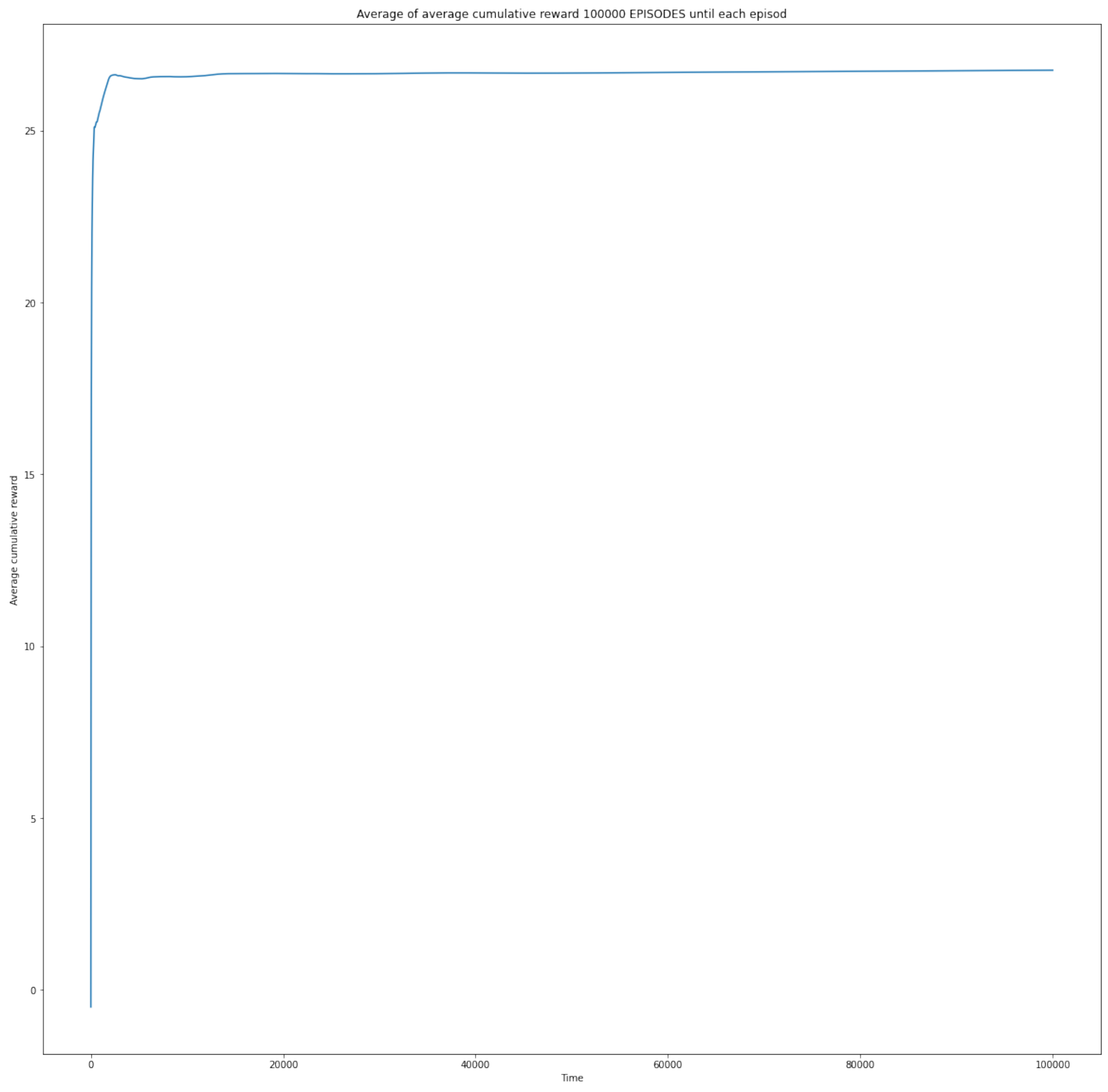
This code is the primary code provided in the assignment. Let see what happens.

[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

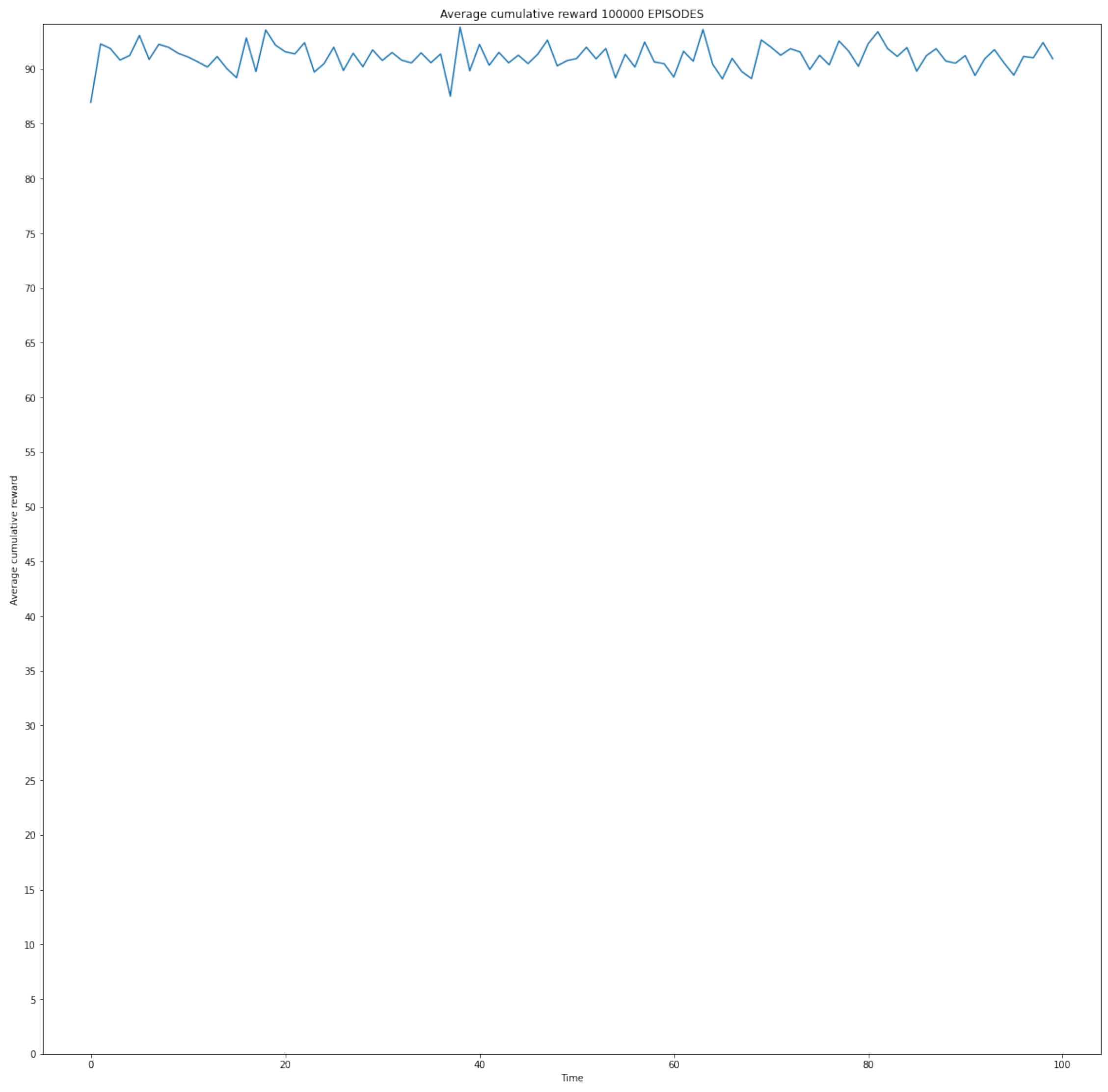


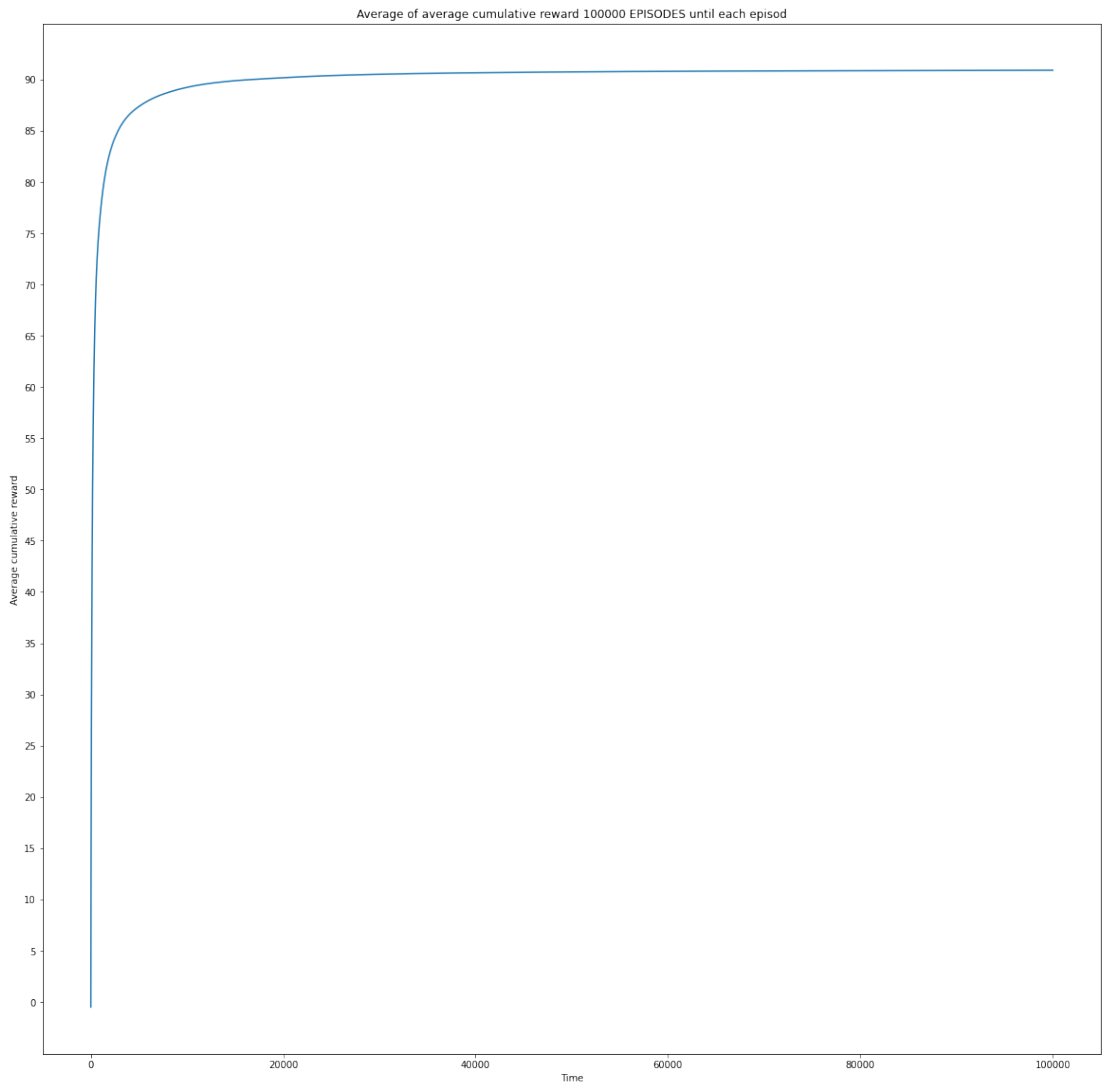
As you can see there is no sign of learning and because there is not any. I also implemented a function for ploting. In this function we average every 1000 point and show them in order to have a more clear plot. ## Add Q learning

[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

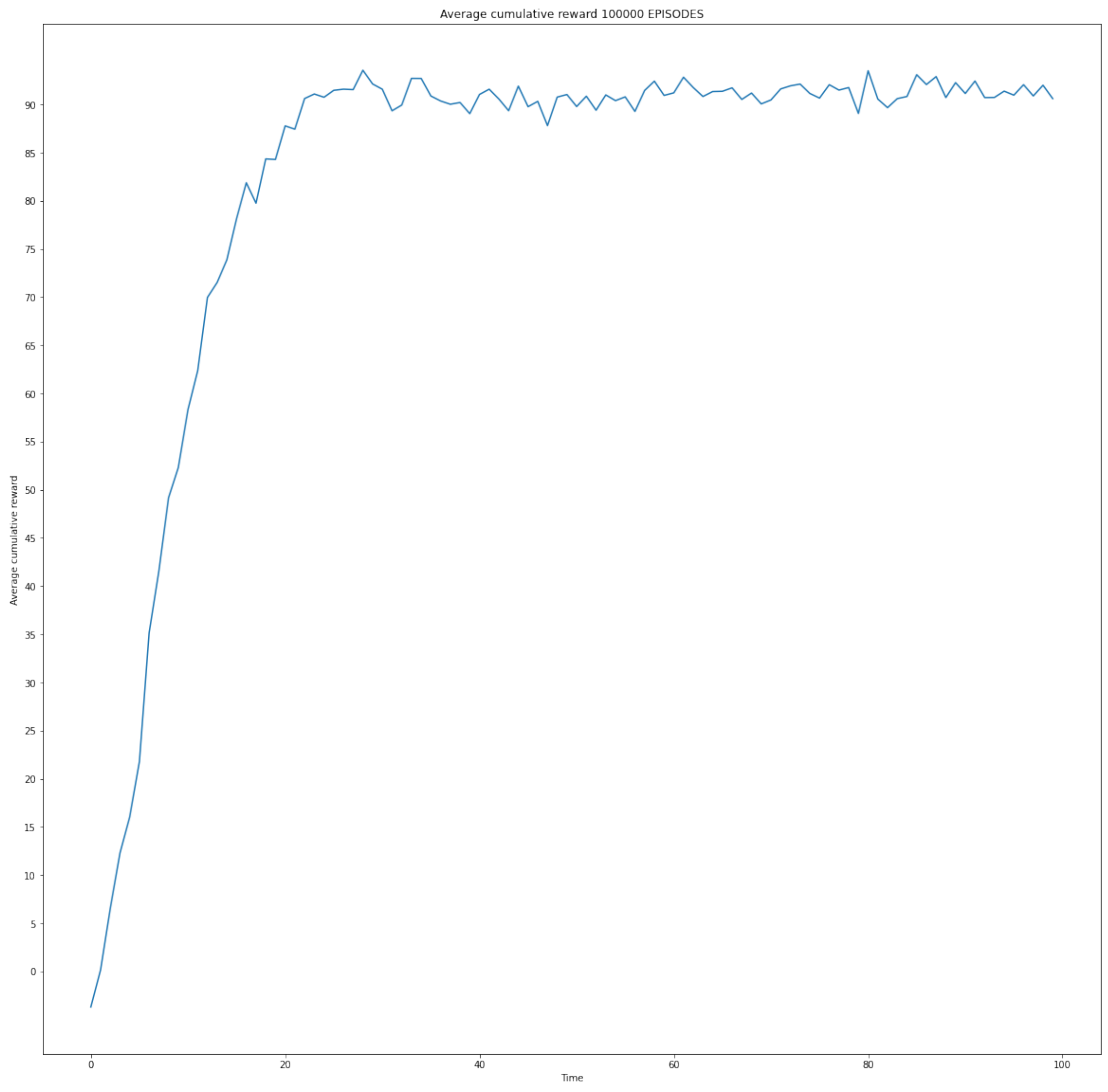


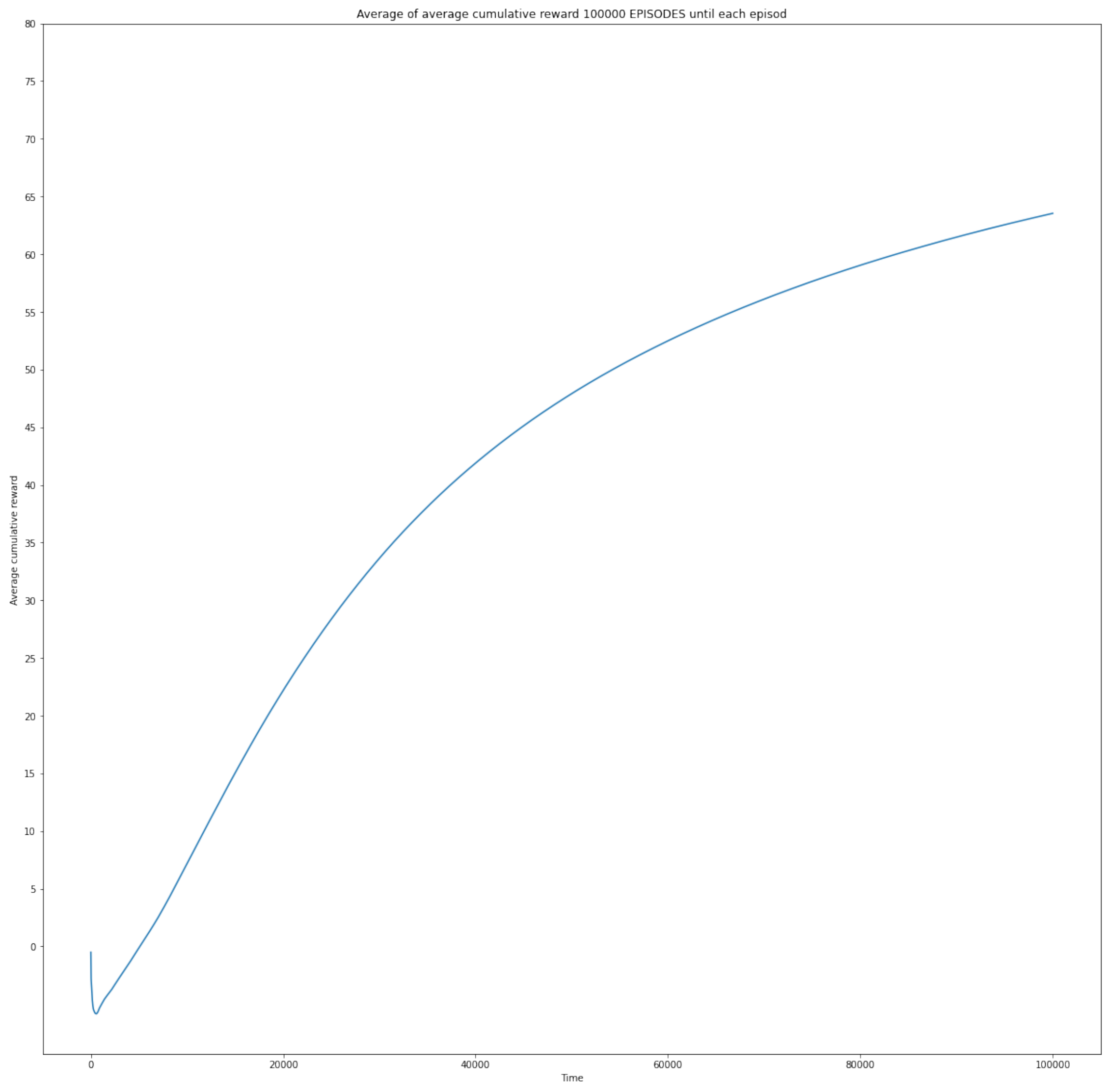
This one is looking better with higher values but because epsilon is very small and does not change the learning is not visible in the plot and also its limited. ## Add Epsilon decay In this part I applied the epsilon decay with DECAY = 0.9999, min_EPSILON = 0.1 and primary EPSILON = 1.

[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
        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.

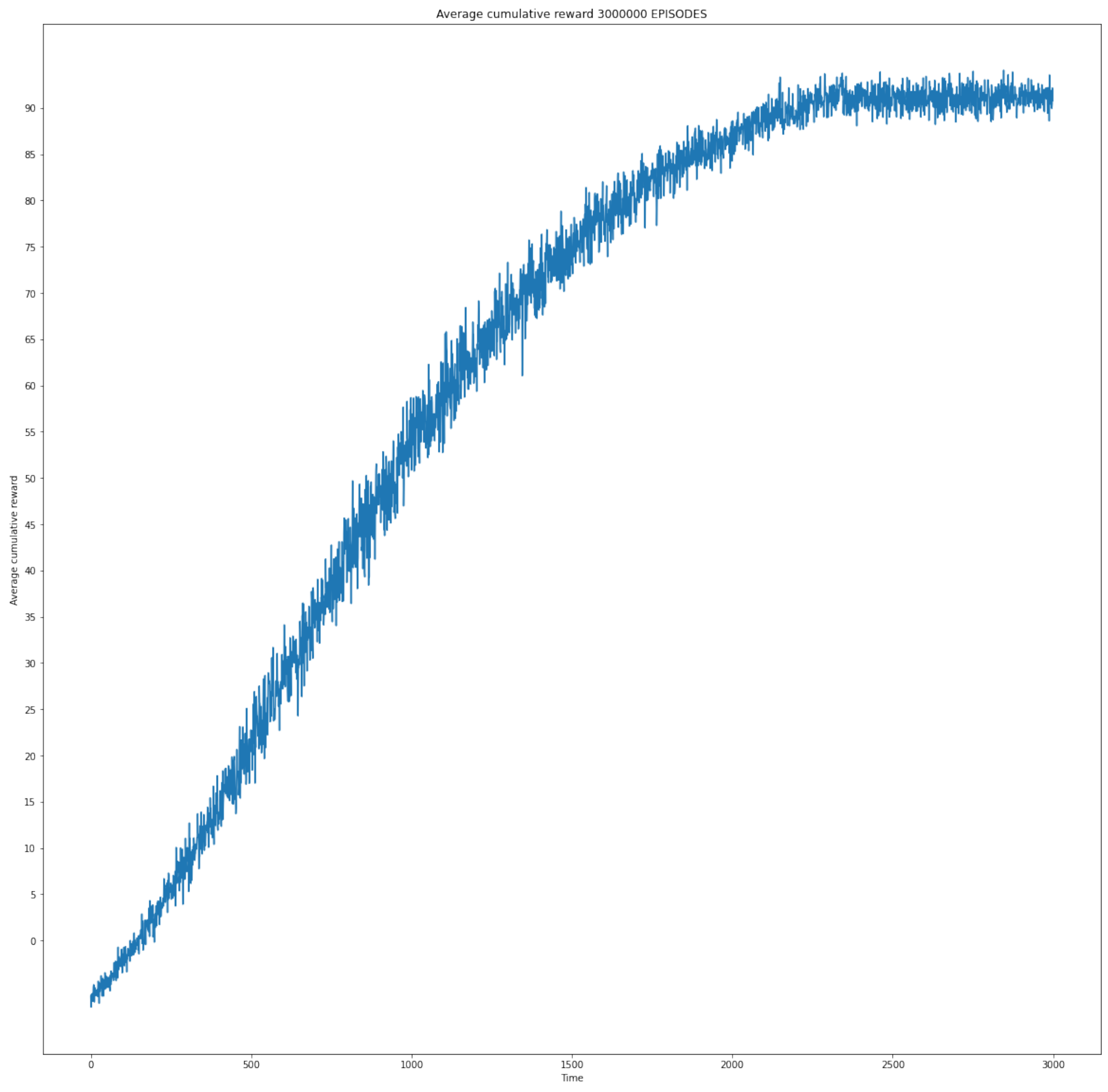
```
[16]: EPISODES = 3000000
      EPSILON = 1
      min_EPSILON = 0.1
      DECAY = 0.999999
      GAMMA = 0.9
      LEARNING_RATE = 0.1
      average_cumulative_rewards = []

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

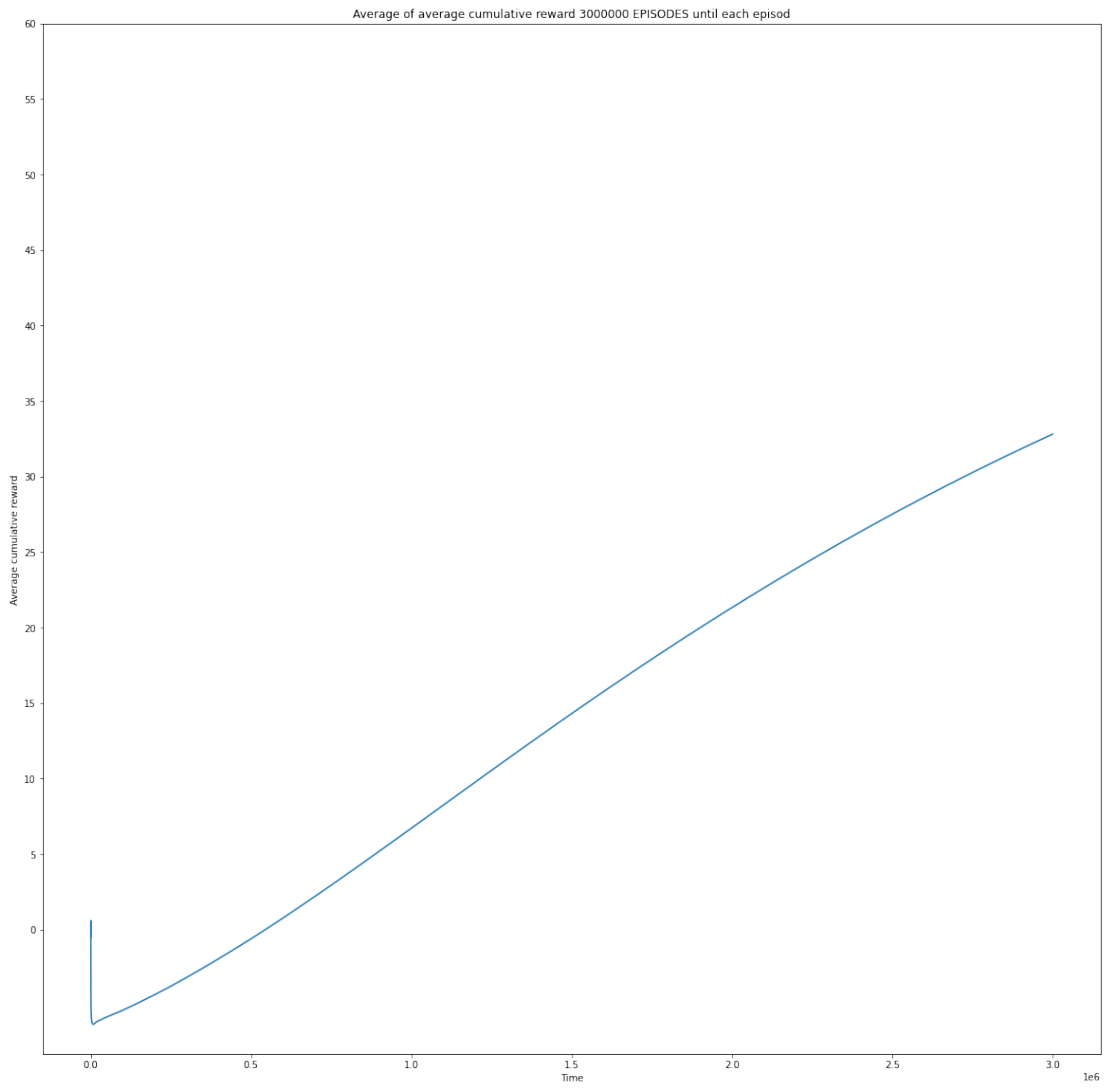
```
def main():
    env = Ice()
    average_cumulative_reward = 0.0

    # 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)
    # 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 0.31981851969006153
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
500
1000
1500
2000
2500
3000



1.2 Conclusion

In conclusion, in my view, epsilon greedy in my view is very simple and works very well for this problem. Epsilon Decay also works very good and better in performance but in my view it is not necessary at all. In the following I also run the epsilon decay and epsilon greedy for a big number of generations but the results where the same.

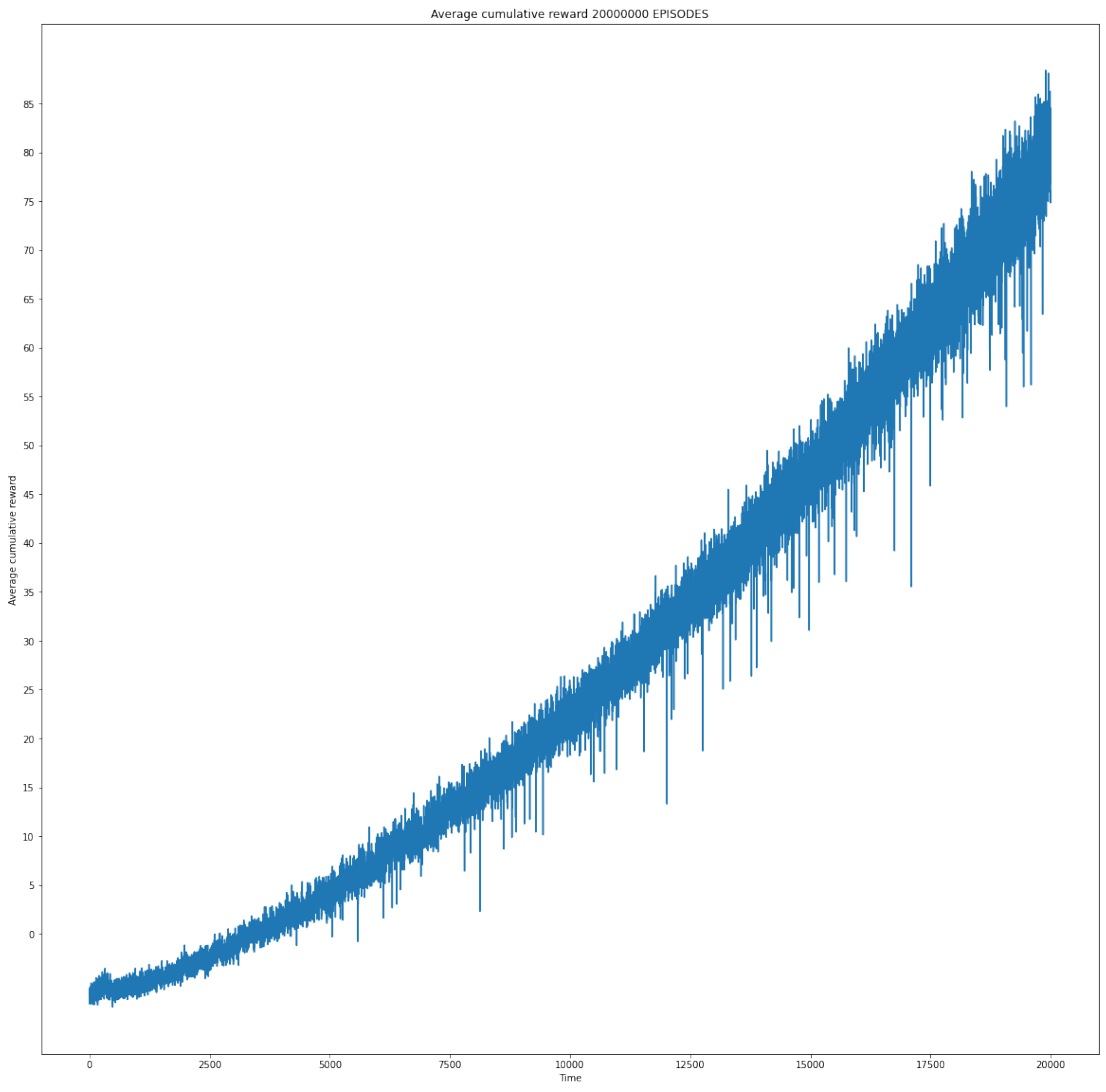
[18]:
```
EPISODES = 20000000
EPSILON = 1
min_EPSILON = 0.1
DECAY = 0.9999999
GAMMA = 0.99
LEARNING_RATE = 0.001
```

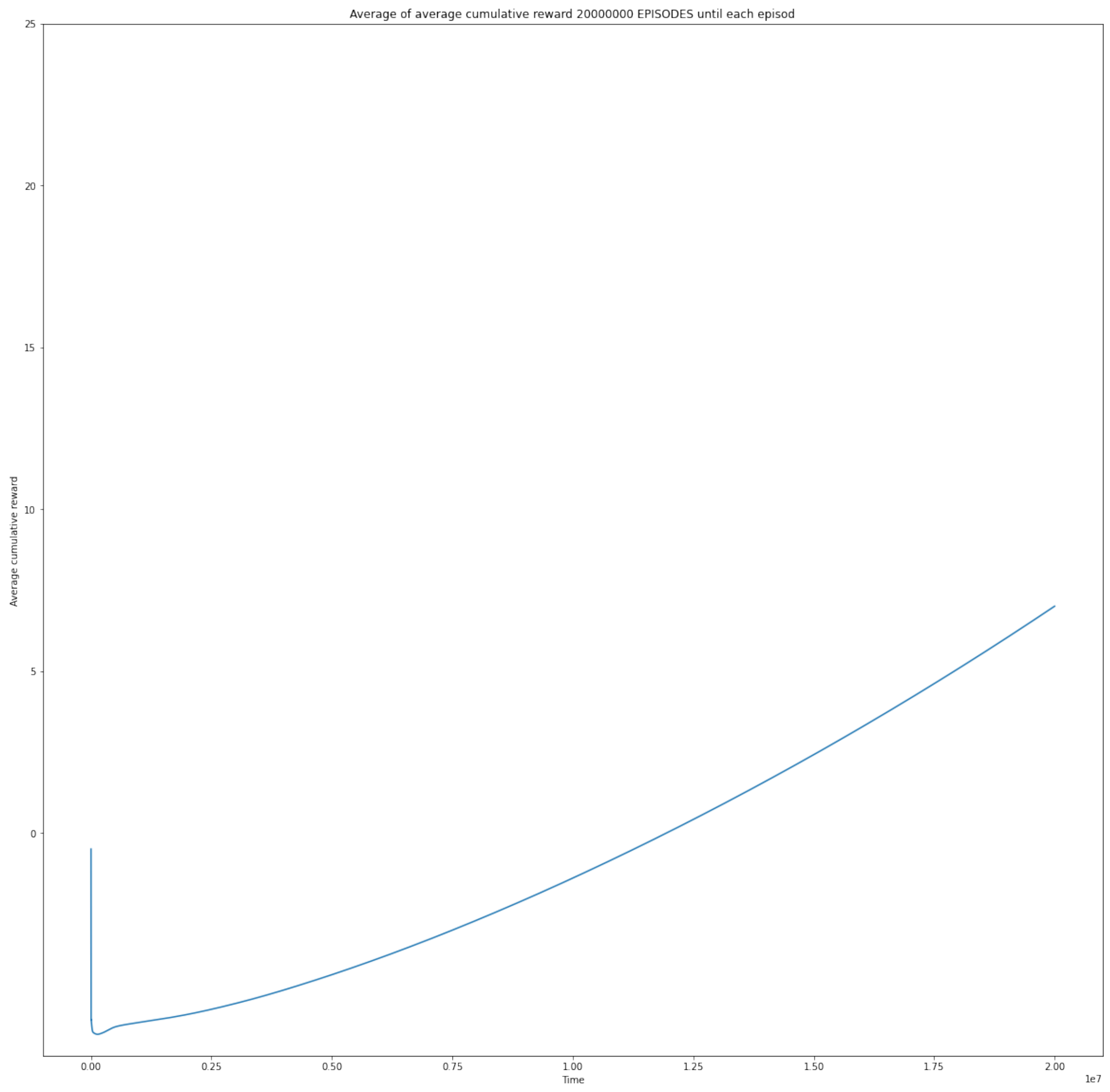
```
average_cumulative_rewards = []

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

        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)

    # 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 0.9999999
200000 -10.0 -8.741821056266987 0.9801985743170336
400000 -10.0 -1.7126888953972066 0.9607893411720586
600000 -10.0 -2.795118933167259 0.9417644366122813
800000 30.0 -5.209369690553468 0.9231162504214226
1000000 -10.0 -4.406327825574534 0.9048373230756607
1200000 -10.0 -2.981880779873464 0.8869203427596255
1400000 -10.0 -7.429438620100973 0.8693581424415379
1600000 -10.0 -5.950903878571323 0.8521436970064615
1800000 10.0 -3.547228751561824 0.835270120445973
2000000 50.0 3.6601814439355245 0.818730663103786
2200000 -10.0 -1.8609520257996353 0.8025187089758151
2400000 -10.0 -4.306665316615627 0.7866277730635841
2600000 10.0 -0.09102152152118603 0.7710514987802367
2800000 -10.0 -2.6499636996852334 0.7557836554077564
3000000 30.0 -2.332278077655987 0.7408181356045965
3200000 10.0 -3.548049932439609 0.7261489529626811
3400000 -10.0 -3.2090920085250443 0.7117702396128426
3600000 -10.0 2.7991501451224368 0.6976762438773969
3800000 50.0 1.1886044877802708 0.6838613279695854
4000000 -10.0 3.400482511076431 0.6703199657383359
4200000 -10.0 5.741824157290883 0.6570467404576289
4400000 -10.0 6.0104258714494065 0.6440363426598346
4600000 -10.0 -0.8315107967327453 0.6312835680118681
4800000 -10.0 10.307886464667355 0.6187833152333198
5000000 -10.0 6.177028752059481 0.60653058405589
```

```
5200000 50.0 1.8459734415459466 0.5945204732232817
5400000 -10.0 3.183684794288618 0.5827481785305337
5600000 -10.0 6.646969317663062 0.5712089909023745
5800000 -10.0 -0.13571928245574427 0.5598982945093992
6000000 10.0 3.9863621106804032 0.5488115649218063
6200000 30.0 6.428131343870209 0.5379443672994882
6400000 -10.0 3.0702373661055637 0.527292354618083
6600000 10.0 9.304471133439232 0.516851265930022
6800000 -10.0 10.550046943879625 0.5066169246602593
7000000 30.0 4.8611690441280455 0.4965852369353801
7200000 -10.0 8.611470283381749 0.4867521899461623
7400000 -10.0 8.288090908688249 0.47711385034229903
7600000 10.0 10.227442945792793 0.4676663626590594
7800000 70.0 15.506103445772501 0.458405947775028
8000000 50.0 12.851355813242641 0.4493289014003959
8200000 30.0 21.992399001335656 0.4404315925952561
8400000 30.0 7.591610292268248 0.4317104623170907
8600000 -10.0 10.583520265096414 0.423162021997148
8800000 150.0 23.38543906723255 0.41478285214500227
9000000 -10.0 16.950471592819607 0.4065696009806328
9200000 -10.0 17.422058037571063 0.3985189830937576
9400000 -10.0 18.64161284217842 0.3906277781295279
9600000 30.0 18.942615942028457 0.38289282950048004
9800000 -10.0 22.945552668068125 0.3753110431236892
10000000 30.0 21.282504796293402 0.3678793861832137
10200000 10.0 33.37891341096056 0.3605948859168944
10400000 90.0 16.330143184357325 0.3534546284272045
10600000 10.0 26.85326890668071 0.34645575751566543
10800000 -10.0 25.289433223281648 0.3395954735403296
11000000 -10.0 21.492214666190357 0.33287103229584275
11200000 -10.0 15.36479646988306 0.32627974391577985
11400000 10.0 29.331680343727296 0.31981897179667024
11600000 -10.0 32.36626299470824 0.3134861315432401
11800000 50.0 35.366893402754975 0.3072786899347011
12000000 10.0 17.91754020605242 0.3011941639114129
12200000 -10.0 31.94666019379376 0.2952301195815736
12400000 10.0 35.42469423370113 0.2893841712477167
12600000 10.0 33.26834386580429 0.2836539804523243
12800000 50.0 26.35910661537538 0.27803725504243854
13000000 -10.0 28.26965546158041 0.2725317482527778
13200000 130.0 51.6292678242071 0.26713525780701836
13400000 -10.0 37.95644058646436 0.26184562503681064
13600000 -10.0 46.31996594811116 0.25666073401830636
13800000 110.0 46.73194783137374 0.25157851072573795
14000000 10.0 32.94821019218621 0.24659692220185128
14200000 30.0 56.54556551873249 0.24171397574461703
14400000 10.0 39.47097548977268 0.23692771811014207
14600000 30.0 32.599707381673 0.23223623473137267
```

```
14800000 -10.0 52.859756938802455 0.22763764895220534
15000000 30.0 58.63742369644292 0.22313012127684095
15200000 170.0 50.81871594346105 0.2187118486339247
15400000 50.0 57.60610309560525 0.21438106365531923
15600000 50.0 35.771912533534206 0.21013603396911498
15800000 50.0 64.78872849076049 0.20597506150666015
16000000 50.0 82.72173810954784 0.20189648182333986
16200000 -10.0 62.48991407210078 0.19789866343272175
16400000 10.0 54.50265523563544 0.1939800071539953
16600000 610.0 87.21548642780209 0.19013894547224636
16800000 -10.0 45.34198255312119 0.18637394191142378
17000000 30.0 49.53817927976512 0.1826834904197647
17200000 230.0 58.30389658379717 0.17906611476732065
17400000 -10.0 51.8814245617992 0.17552036795545128
17600000 30.0 81.03878141410344 0.17204483163801396
17800000 90.0 76.27141875715057 0.1686381155539982
18000000 10.0 47.20534227580813 0.16529885697142027
18200000 10.0 58.888739573784164 0.16202572014219216
18400000 10.0 64.5688198629889 0.1588173957678012
18600000 190.0 65.16409766914786 0.15567260047559595
18800000 -10.0 53.781907224367544 0.15259007630539986
19000000 -10.0 89.63250243012786 0.14956859020633523
19200000 110.0 71.70113611732586 0.1466069335435417
19400000 70.0 95.47477853338262 0.14370392161476403
19600000 230.0 98.14092610365147 0.1408583931763942
19800000 70.0 80.6652384635236 0.13806920997900352
238.327 234.721 243.234 0.000
240.439 0.000 254.339 0.000
241.775 255.916 253.134 0.000
238.198 0.000 0.000 0.000
```


Average cumulative reward 20000000 EPISODES
Average cumulative reward
Time
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
0
2500
5000
7500
10000
12500
15000
17500
20000

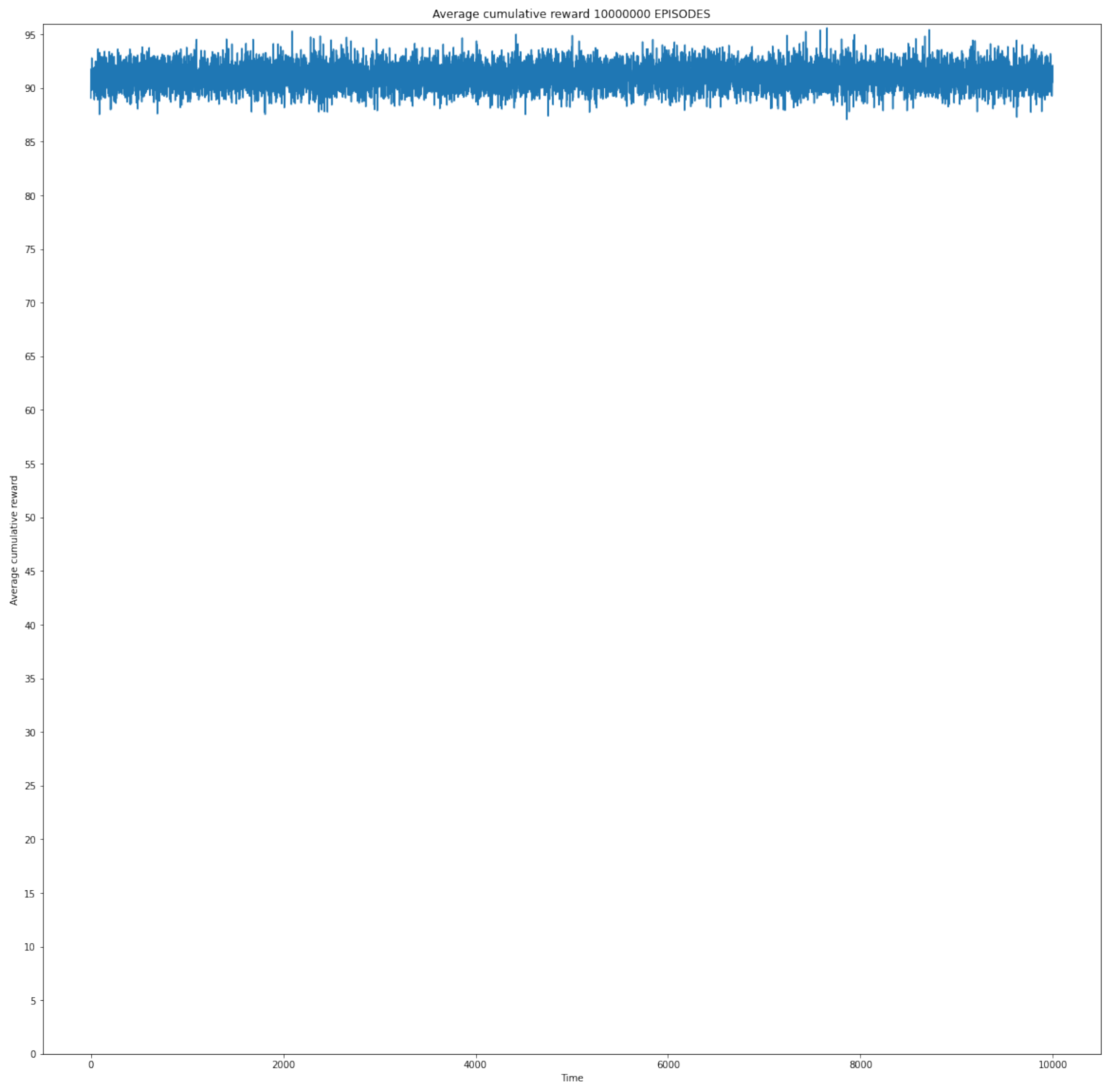


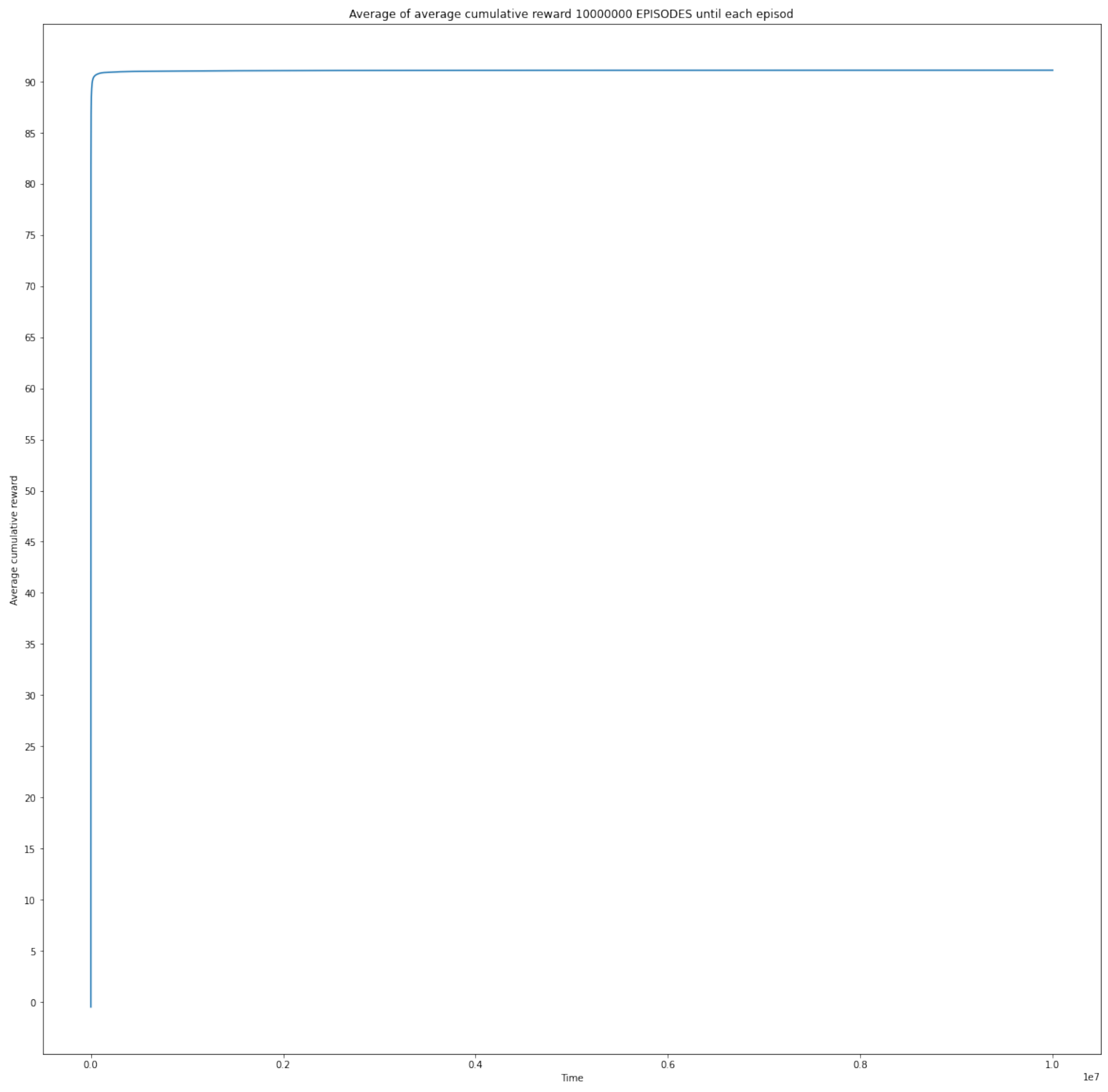
[21]:
```
EPISODES = 10000000
EPSILON = 0.1
GAMMA = 0.9
LEARNING_RATE = 0.1
average_cumulative_rewards = []

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 / 100) == 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
100000 100.0 103.93051730523926 0.1
200000 100.0 82.3716944674481 0.1
300000 100.0 75.84911605619274 0.1
400000 100.0 84.36126252120249 0.1
500000 100.0 92.99323637089128 0.1
600000 100.0 84.06193594881591 0.1
700000 100.0 91.92139547209658 0.1
800000 100.0 83.9574126650045 0.1
900000 100.0 88.20158706349126 0.1
1000000 100.0 88.8601078240861 0.1
1100000 100.0 94.82879829697912 0.1
1200000 100.0 95.67834379107987 0.1
1300000 -10.0 91.05615056332726 0.1
1400000 100.0 86.47257890005534 0.1
1500000 100.0 86.9800919254992 0.1
1600000 100.0 95.87521627591029 0.1
1700000 100.0 85.98917024295888 0.1
1800000 100.0 95.1537609595085 0.1
1900000 100.0 92.45328200365343 0.1
2000000 100.0 87.28583602960316 0.1
2100000 100.0 95.90722381074721 0.1
2200000 10.0 94.50537337928436 0.1
2300000 100.0 90.46593566269306 0.1
2400000 -10.0 75.42467321541909 0.1
2500000 100.0 88.20681249513096 0.1
2600000 100.0 95.64305920271619 0.1
2700000 -10.0 85.6146865126558 0.1
2800000 100.0 74.35407585705472 0.1
2900000 120.0 96.34016718890021 0.1
3000000 100.0 90.08783891691817 0.1
3100000 100.0 96.44480637251371 0.1
3200000 140.0 94.38471770616425 0.1
3300000 100.0 84.85559847213274 0.1
3400000 120.0 90.06340492340894 0.1
3500000 120.0 100.95655857027585 0.1
3600000 100.0 90.41709375115875 0.1
3700000 100.0 93.63004023935184 0.1
```

```
3800000 100.0 94.19694180017025 0.1
3900000 120.0 100.97974961810617 0.1
4000000 100.0 92.05580279854982 0.1
4100000 120.0 89.76235407812928 0.1
4200000 100.0 91.24090106319007 0.1
4300000 10.0 90.6112655356371 0.1
4400000 100.0 92.54648319104965 0.1
4500000 100.0 98.8212525780498 0.1
4600000 100.0 84.29614994697428 0.1
4700000 100.0 96.27173919312555 0.1
4800000 120.0 93.11574820930723 0.1
4900000 100.0 93.27796758995245 0.1
5000000 100.0 91.78287604912461 0.1
5100000 120.0 90.74092354183152 0.1
5200000 100.0 90.65354802033086 0.1
5300000 120.0 93.97444856822933 0.1
5400000 100.0 79.99349251568086 0.1
5500000 120.0 93.3634388425707 0.1
5600000 -10.0 93.27151222183582 0.1
5700000 120.0 102.61221415534317 0.1
5800000 120.0 98.11385337634538 0.1
5900000 100.0 95.53018660701353 0.1
6000000 100.0 87.19001122689214 0.1
6100000 100.0 92.76901770340075 0.1
6200000 100.0 97.35995462215553 0.1
6300000 100.0 90.26907297572369 0.1
6400000 100.0 90.23215248459812 0.1
6500000 100.0 78.06148944964566 0.1
6600000 100.0 95.15839429697758 0.1
6700000 120.0 86.42413293291456 0.1
6800000 100.0 81.41274310449445 0.1
6900000 100.0 95.21824849150373 0.1
7000000 100.0 99.33392512205151 0.1
7100000 160.0 81.12777734739957 0.1
7200000 100.0 90.14871868399159 0.1
7300000 100.0 94.87518781965466 0.1
7400000 120.0 89.18377458306931 0.1
7500000 100.0 94.1039948915198 0.1
7600000 120.0 93.38505312084166 0.1
7700000 120.0 90.2448822153692 0.1
7800000 100.0 88.03151843343795 0.1
7900000 120.0 88.94605286913617 0.1
8000000 100.0 98.22055246620351 0.1
8100000 100.0 91.41928718048348 0.1
8200000 100.0 75.62025086968241 0.1
8300000 -10.0 78.94943103683654 0.1
8400000 100.0 92.99697488967051 0.1
8500000 100.0 93.94946960799984 0.1
```

```
8600000 100.0 98.31002872178985 0.1
8700000 100.0 93.82756429043629 0.1
8800000 100.0 94.6798600606303 0.1
8900000 100.0 96.55873851881691 0.1
9000000 100.0 95.7946180256268 0.1
9100000 120.0 98.61577351043711 0.1
9200000 100.0 78.31742132050347 0.1
9300000 100.0 90.94608405582089 0.1
9400000 100.0 98.73054238225124 0.1
9500000 100.0 91.62735408591719 0.1
9600000 100.0 92.5969768386569 0.1
9700000 100.0 98.34214649029363 0.1
9800000 100.0 90.03875214257664 0.1
9900000 120.0 91.00909951848777 0.1
81.792 90.803 100.000 0.000
73.796 0.000 90.000 0.000
66.705 83.942 81.018 0.000
61.646 0.000 0.000 0.000
```


Average cumulative reward 10000000 EPISODES
Average cumulative reward
Time
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
0
2000
4000
6000
8000
10000


Average of average cumulative reward 10000000 EPISODES until each episod
Average cumulative reward
Time
1e7
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
0.0
0.2
0.4
0.6
0.8
1.0

```
[ ]:
```