

# OpenAI Gym Taxi

Matthew Johnson

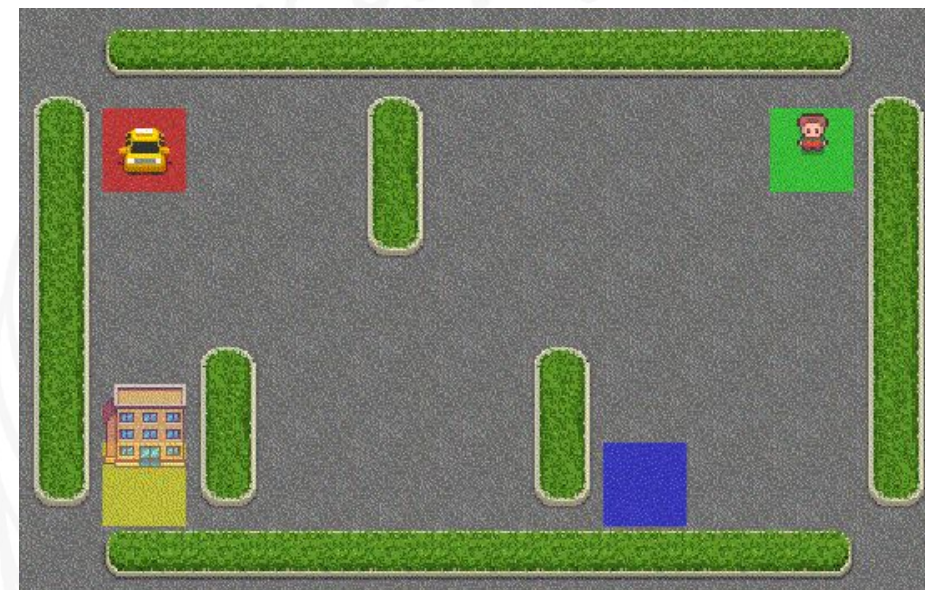
CSE 368

12/1/22



# Project Description

This project is built off of the “Taxi-v3” environment which is one of the OpenAI Toy Text environments. The environment is a grid world with a car as the agent. The grid world consists of four different color tiles that the car, passenger, and destination can all randomly spawn on. The goal is for the car to pick up the passenger and deliver them to the destination.



Source: [https://www.gymnasium.dev/environments/toy\\_text/taxi/](https://www.gymnasium.dev/environments/toy_text/taxi/)

# Background

The environment has all the necessary parts to be solved in a tabular method.

The environment is broken down into 500 possible states between all of the possible taxi, passenger, and destination locations.

Additionally there are 6 possible actions.

These states and actions can be used to train the taxi through reinforcement learning.

## Actions

There are 6 discrete deterministic actions:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

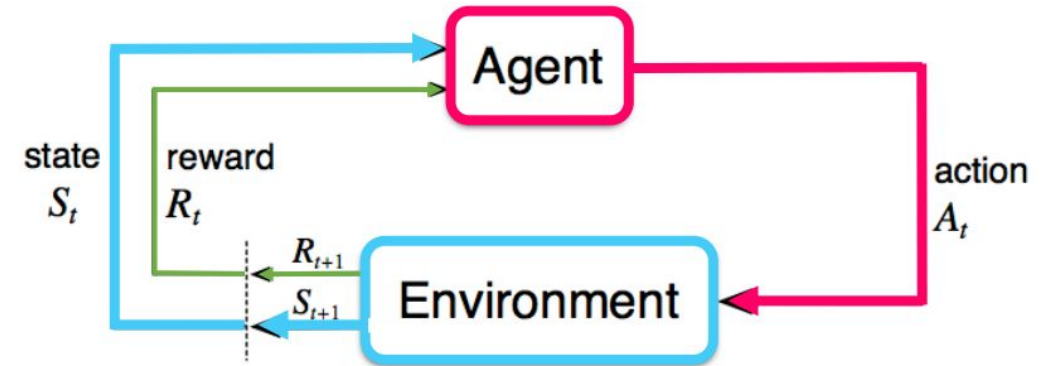
Source: [https://www.gymnasium.dev/environments/toy\\_text/taxi/](https://www.gymnasium.dev/environments/toy_text/taxi/)

## Background Continued

There are three algorithms that were used to train the taxi:

- SARSA
- Q-Learning
- Double Q-Learning

These algorithms can have their parameters tuned in order to give a variety of results that lead to the problem converging.

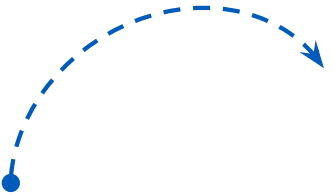


Source: [https://piazza.com/class\\_profile/get\\_resource/l79ev0g88vc7a7/l9rk6yp1fic2hd](https://piazza.com/class_profile/get_resource/l79ev0g88vc7a7/l9rk6yp1fic2hd)



# Implementation

In order to choose actions in any given state there must be a policy to follow. A  $\epsilon$ -greedy policy was implemented and used for all of three algorithms.



**NOTE:** Double Q-Learning differs from SARSA and Q-Learning

```
def get_action(env, qtable, state, epsilon):
    policy = np.random.uniform(0,1)
    if policy > epsilon:
        return np.argmax(qtable[state,:])
    else:
        return env.action_space.sample()

def Double_Q_get_action(env, qtableA, qtableB, state, epsilon):
    policy = np.random.uniform(0,1)
    if policy > epsilon:
        tableSums = []
        x=0
        for x in range(6):
            tableSums.append(qtableA[state,x]+qtableB[state,x])

        maxVal = max(tableSums)
        maxList = []
        for x in range(6):
            if tableSums[x] == maxVal:
                maxList.append(x)
        action = np.random.choice(maxList)
        return action
    else:
        return env.action_space.sample()
```

# Implementation Continued

The implementation of SARSA and Q-Learning are very similar, and only have a few small changes. While Double Q-learning is also similar it uses a bit of a different approach than the other two.

## Double Q-Learning

```
randomUpdate = np.random.uniform(0,1)
if randomUpdate > .5:
    action2 = get_table(qtableA, state2)
    qtableA[state,action] = qtableA[state,action] + learning_rate * \
        (reward + discount_factor * qtableB[state2,action2]-qtableA[state,action])
else:
    action2 = get_table(qtableB, state2)
    qtableB[state,action] = qtableB[state,action] + learning_rate * \
        (reward + discount_factor * qtableA[state2,action2]-qtableB[state,action])
```

## SARSA

```
qtable[state,action] = qtable[state,action] + learning_rate * \
    (reward + discount_factor * qtable[state2,action2]-qtable[state,action])
```

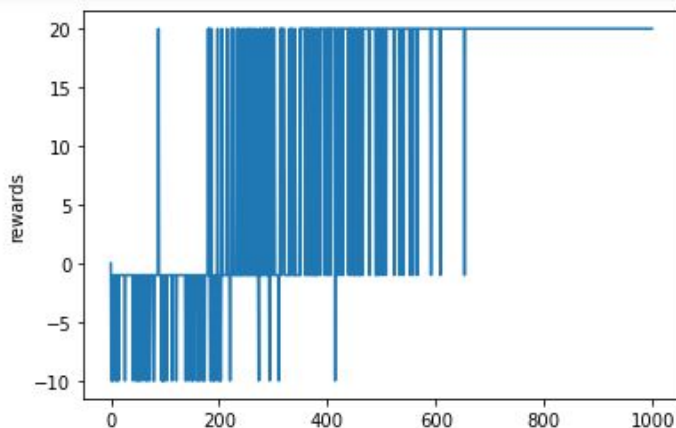
## Q-Learning

```
qtable[state,action] = qtable[state,action] + learning_rate * \
    (reward + discount_factor * np.max(qtable[state2,:])-qtable[state,action])
```

# Results - Parameter Tuning

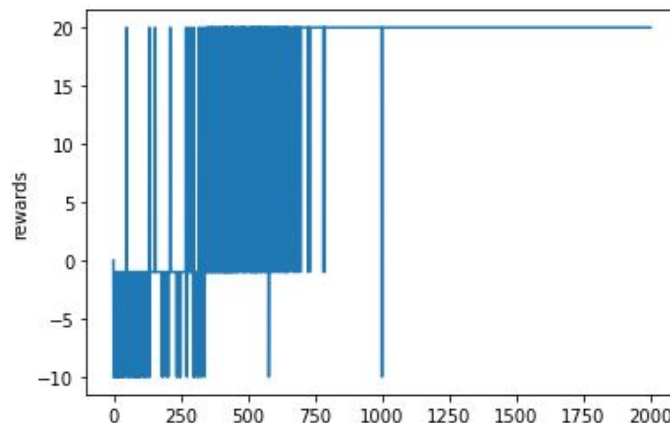
## Optimal Q-Learning parameters:

episodes: 1000  
timesteps: 50  
epsilon: 1  
epsilon decay: .005  
learning rate: .9  
discount factor: 1



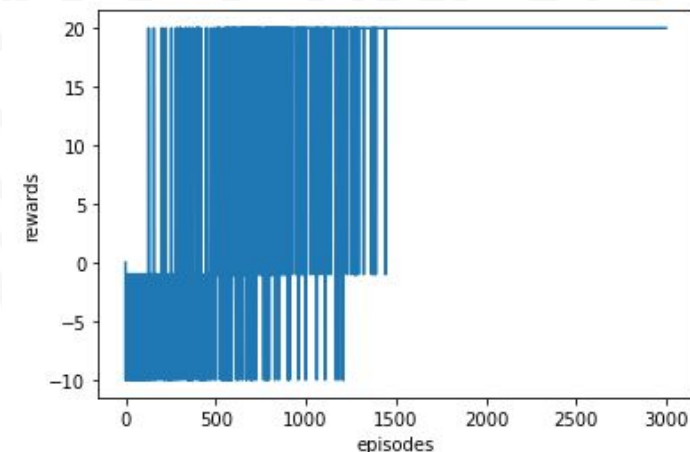
## Optimal SARSA parameters:

episodes: 2000  
timesteps: 50  
epsilon: .4  
epsilon decay: .005  
learning rate: .5  
discount factor: .95



## Optimal Double Q-Learning parameters:

episodes: 3000  
timesteps: 50  
epsilon: 1  
epsilon decay: .005  
learning rate: 1  
discount factor: 1

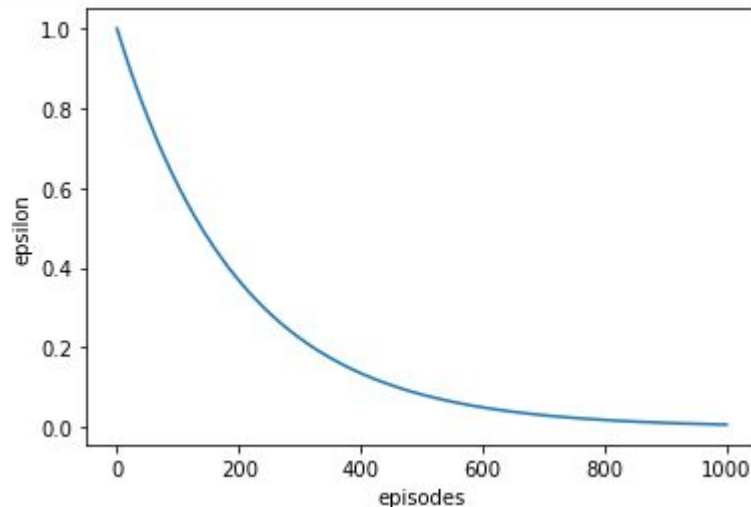




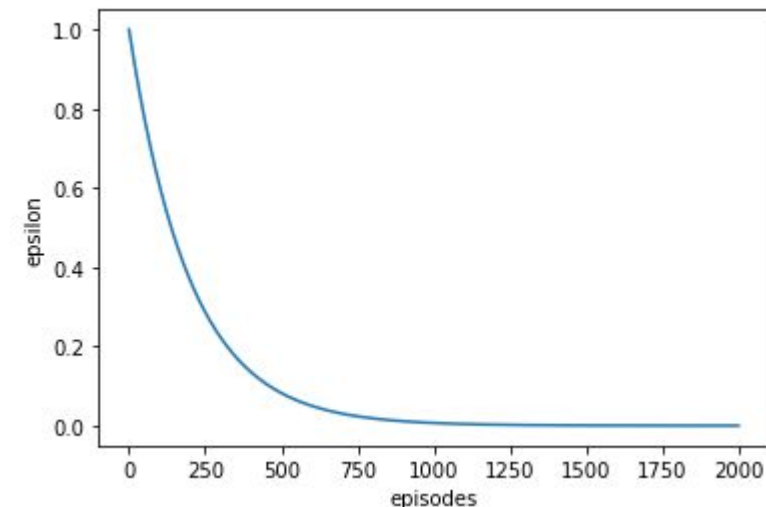
# Results - Epsilon

The epsilon and its decay varies in how each algorithm is implemented. Each algorithm has a different amount of time for its exploration and exploitation.

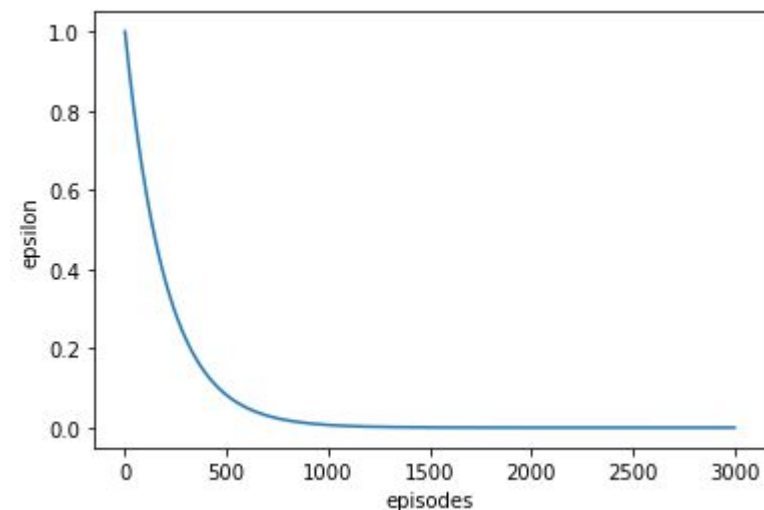
**Q-Learning**



**SARSA**



**Double Q-Learning**





## Results - Environment Simulation



## Summary

The taxi agent reached its goal using SARSA, Q-Learning, and Double Q-Learning. They were most effective in the order of Q-learning, SARSA, and then Double Q-learning.



Thank You For  
Listening

# References

- [https://www.gymlibrary.dev/environments/toy\\_text/taxi/](https://www.gymlibrary.dev/environments/toy_text/taxi/)
- [https://piazza.com/class\\_profile/get\\_resource/l79ev0g88vc7a7/l9rk6yp1fic2hd](https://piazza.com/class_profile/get_resource/l79ev0g88vc7a7/l9rk6yp1fic2hd)

