Microsoft Student Partners

RL Exercise
: Cartpole (OpenAI GYM)

Presentation Title Subhead

01001101
01010011
01010000
CODE IS LIFE

Microsoft

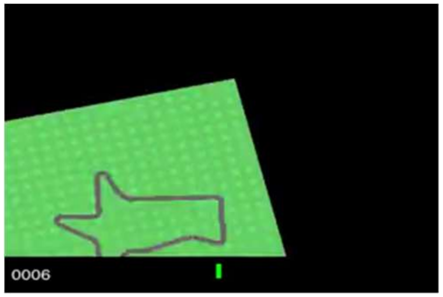
What is GYM?

Open AI: Python module named ‘GYM’

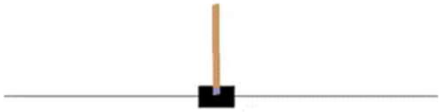
- Provides various RL-able environments

Boxing
(Atari)

Breakout
(Atari)



CarRacing
(Box2d)

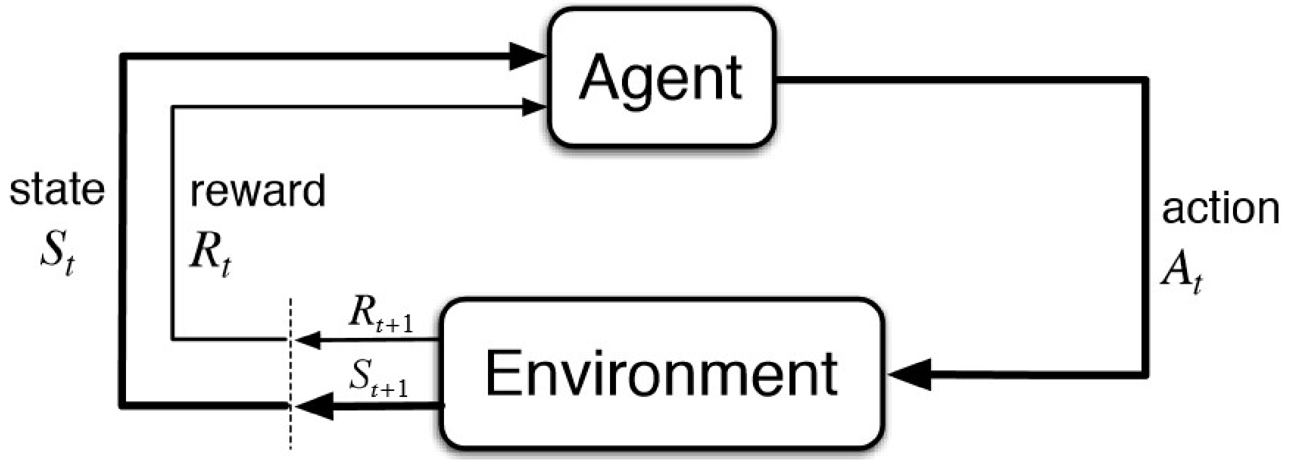


Cartpole
(Classic)

What is GYM?

Reinforcement Learning

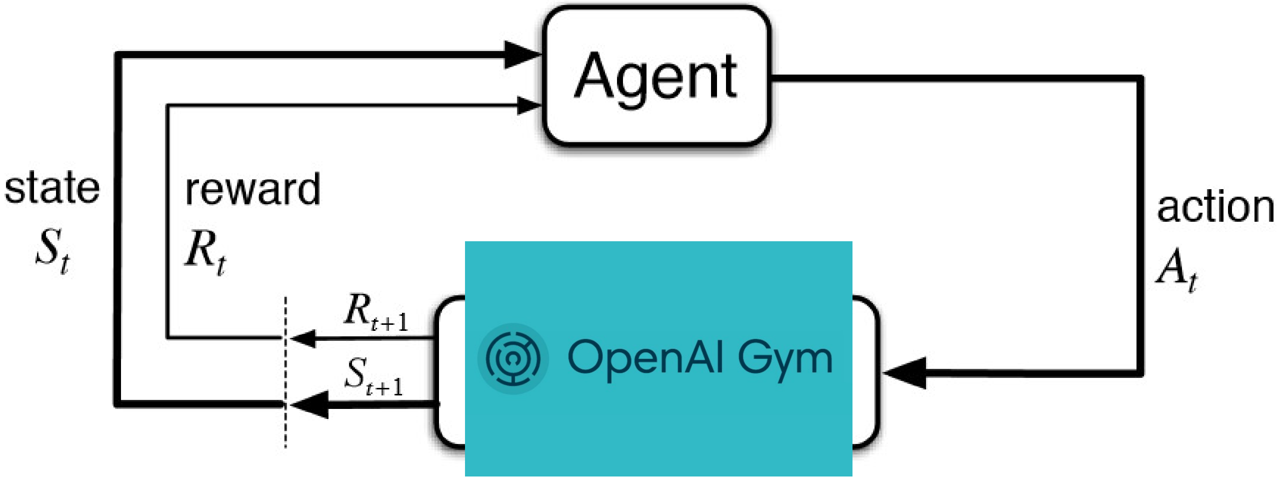
- Learning between Agent-Environment communication



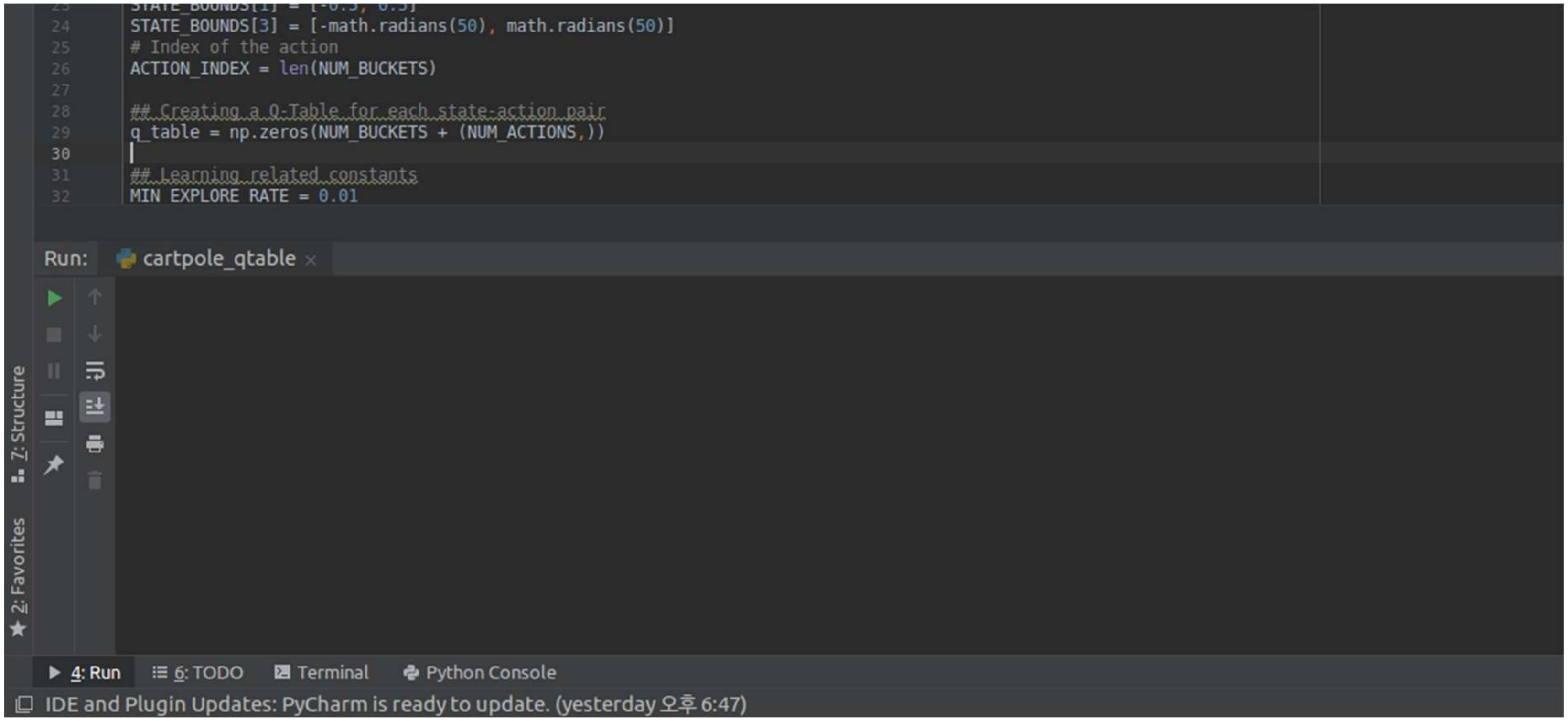
What is GYM?

Reinforcement Learning

- Learning between Agent-Environment communication



Training Cartpole-solver: Video



```
STATE_BOUNDS[1] = [-0.5, 0.5]
STATE_BOUNDS[3] = [-math.radians(50), math.radians(50)]
# Index of the action
ACTION_INDEX = len(NUM_BUCKETS)

## Creating a Q-Table for each state-action pair
q_table = np.zeros(NUM_BUCKETS + (NUM_ACTIONS,))

## Learning related constants
MIN_EXPLORE_RATE = 0.01
```

IDE and Plugin Updates: PyCharm is ready to update. (yesterday 오후 6:47)

Definition of State, Action, Reward

State

- Position, Radian, ...

Action

- Left or Right Move

Reward

- Time alive (=Step)

Done

Quiz.

1. Q-Table RL은 Action space와 State space가 _______ 때 사용한다.

2. 한 번의 Simulation은 여러 번의 _______로 구성된다. episode

3. 한 번의 Episode는 여러 번의 _______으로 구성된다. step

4. Q-Table의 크기는 _______ x _______이다. action의 갯수 X sate의 갯수

   Q-value

5. Action은 Q-Table에서 주어진 State에 해당하는 Entry중 가장 _______가 큰 Action으로 결정된다.

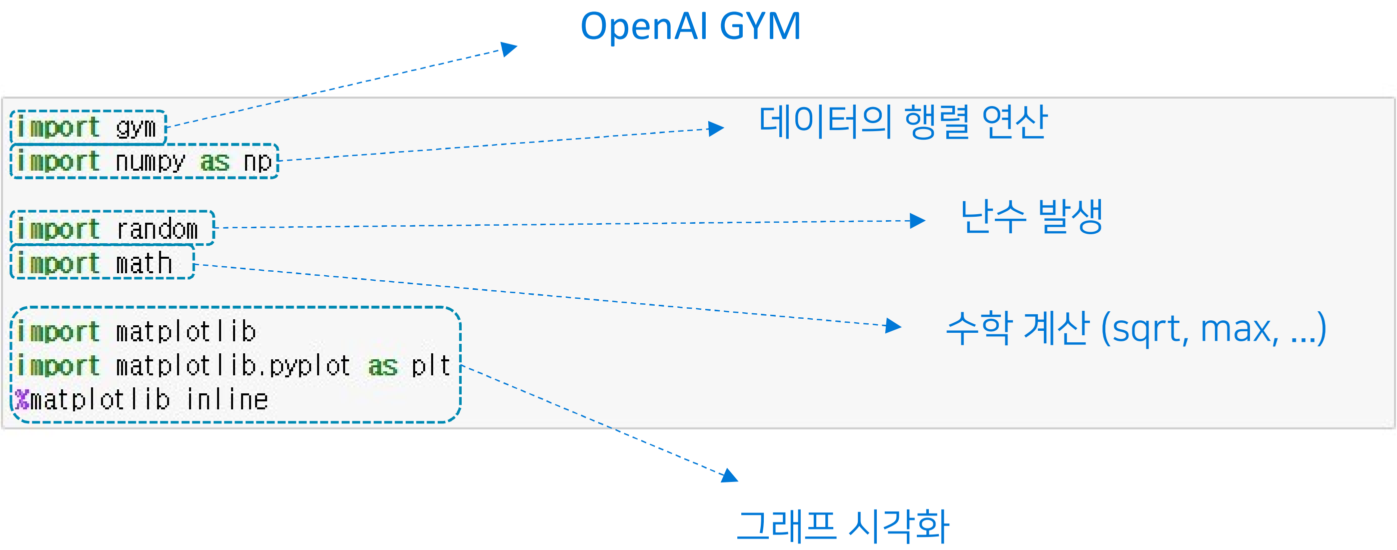
   Learning Discount 상이

6. _______ rate, _______ rate, _______ factor은 Episode별로 _____하다.

DIVE IN TO THE CODE

Environment Settings

Import Modules



```
import gym
import numpy as np

import random
import math

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

OpenAI GYM

데이터의 행렬 연산

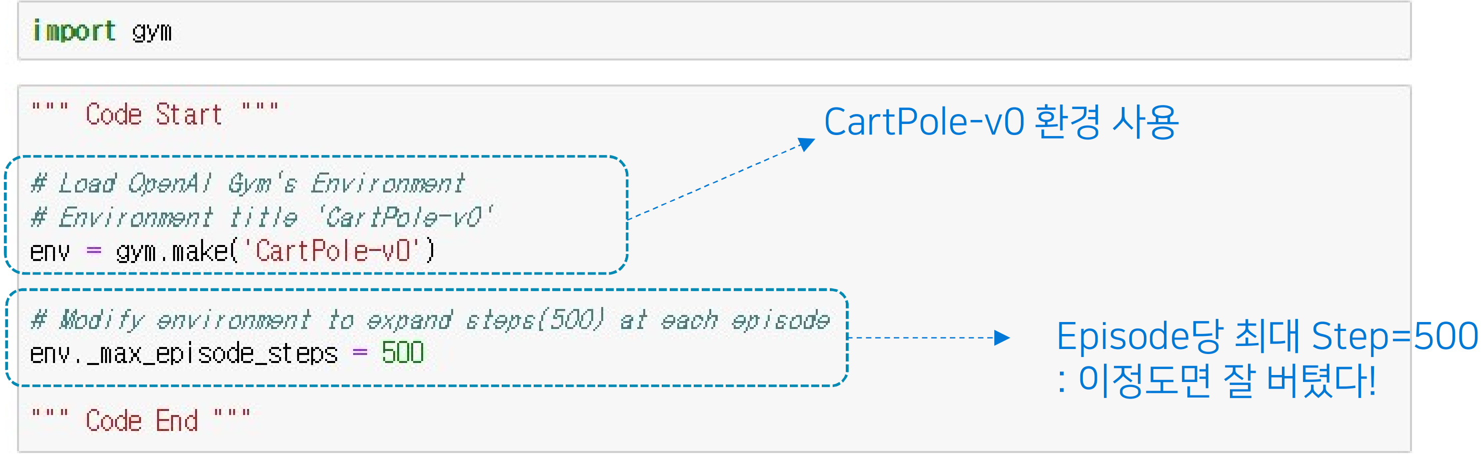
난수 발생

수학 계산 (sqrt, max, ...)

그래프 시각화

Environment Settings

Load OpenAI Gym



```
import gym
```

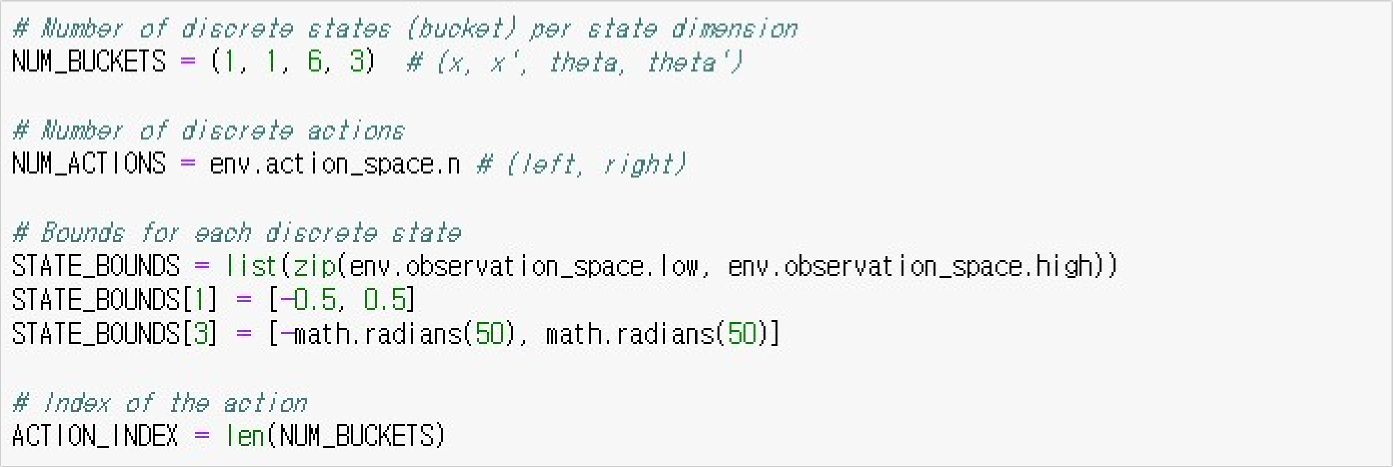
```
""" Code Start """

# Load OpenAI Gym's Environment
# Environment title 'CartPole-v0'
env = gym.make('CartPole-v0')

# Modify environment to expand steps(500) at each episode
env._max_episode_steps = 500

""" Code End """
```

CartPole-v0 환경 사용

Episode당 최대 Step=500
: 이정도면 잘 버텼다!

Environment Settings

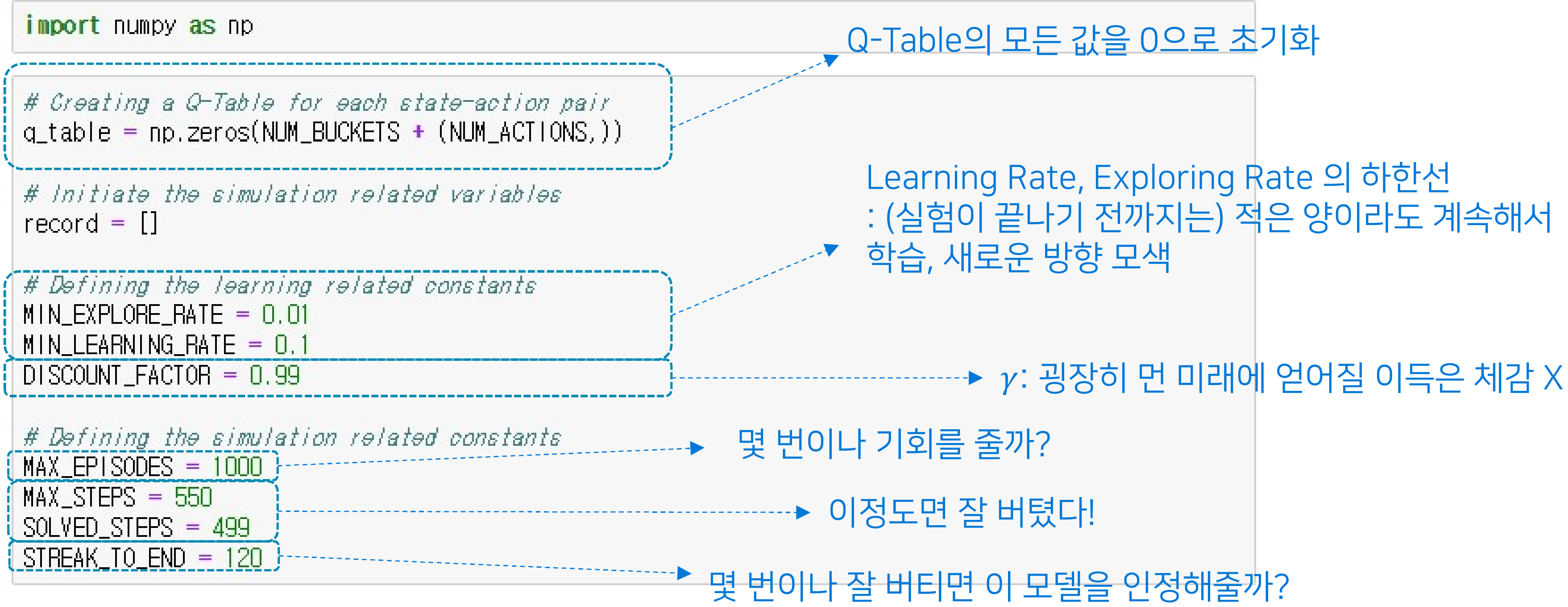
Defining the environment related constants



```
# Number of discrete states (bucket) per state dimension
NUM_BUCKETS = (1, 1, 6, 3)  # (x, x', theta, theta')

# Number of discrete actions
NUM_ACTIONS = env.action_space.n # (left, right)

# Bounds for each discrete state
STATE_BOUNDS = list(zip(env.observation_space.low, env.observation_space.high))
STATE_BOUNDS[1] = [-0.5, 0.5]
STATE_BOUNDS[3] = [-math.radians(50), math.radians(50)]

# Index of the action
ACTION_INDEX = len(NUM_BUCKETS)
```

Environment Settings

Defining the Q-learning related constants



```
import numpy as np
```

```
# Creating a Q-Table for each state-action pair
q_table = np.zeros(NUM_BUCKETS + (NUM_ACTIONS,))

# Initiate the simulation related variables
record = []

# Defining the learning related constants
MIN_EXPLORE_RATE = 0.01
MIN_LEARNING_RATE = 0.1
DISCOUNT_FACTOR = 0.99

# Defining the simulation related constants
MAX_EPISODES = 1000
MAX_STEPS = 550
SOLVED_STEPS = 499
STREAK_TO_END = 120
```

Q-Table의 모든 값을 0으로 초기화

Learning Rate, Exploring Rate 의 하한선
: (실험이 끝나기 전까지는) 적은 양이라도 계속해서 학습, 새로운 방향 모색

γ: 굉장히 먼 미래에 얻어질 이득은 체감 X

몇 번이나 기회를 줄까?

이정도면 잘 버텼다!

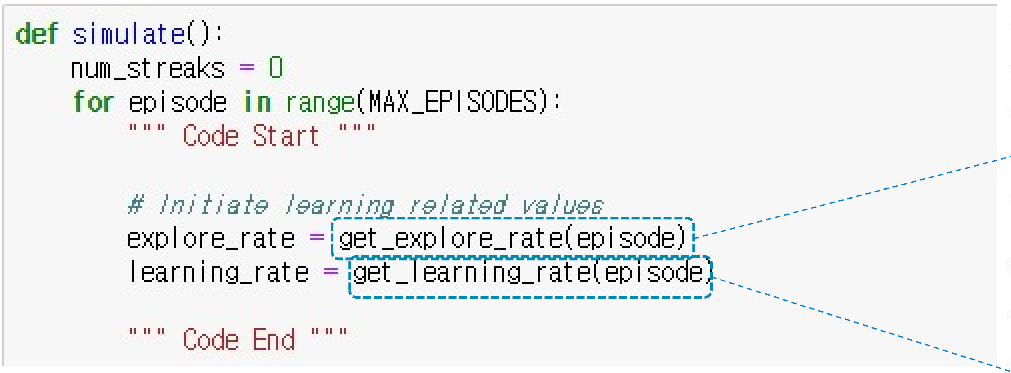
몇 번이나 잘 버티면 이 모델을 인정해줄까?

Microsoft Student Partners

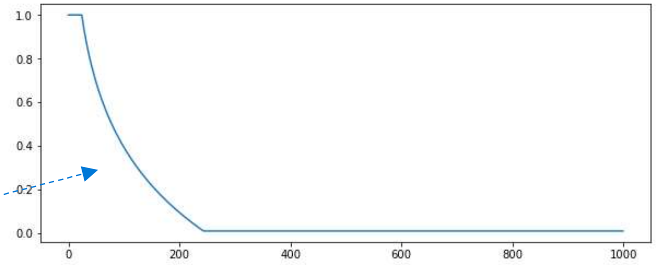
Microsoft

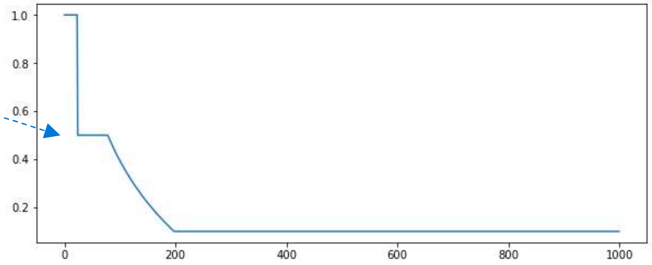
Simulation

Get exploring rate & learning rate depend on episode.#



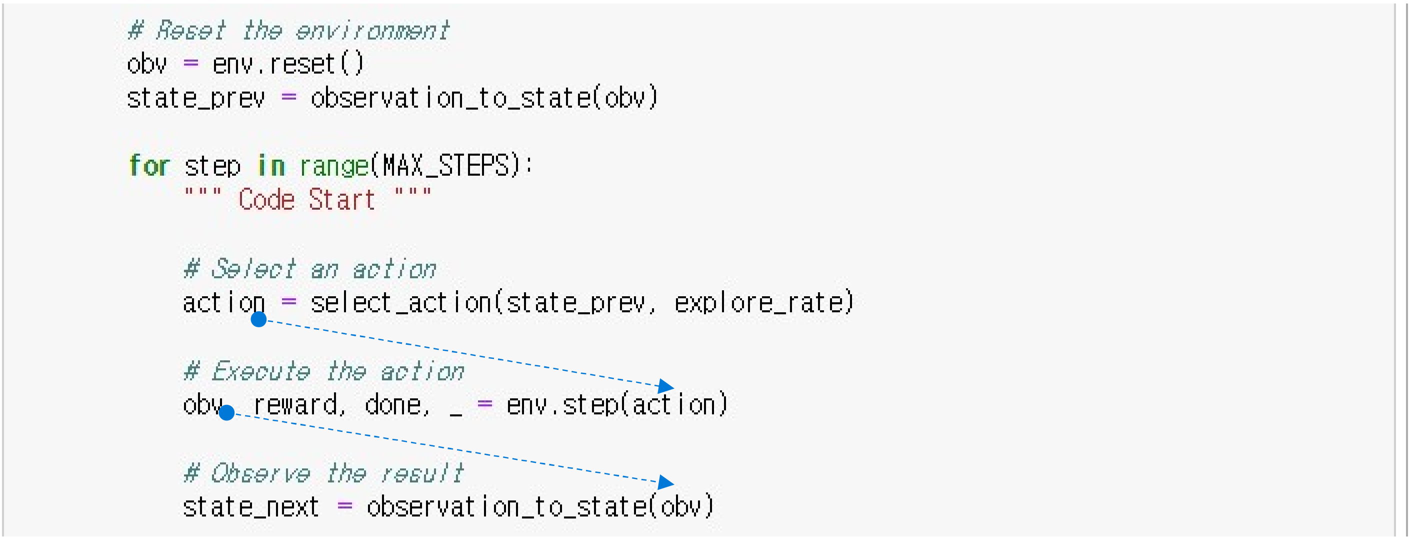
```
def simulate():
    num_streaks = 0
    for episode in range(MAX_EPISODES):
        """ Code Start """

        # Initiate learning related values
        explore_rate = get_explore_rate(episode)
        learning_rate = get_learning_rate(episode)

        """ Code End """
```





Simulation

At each episode...

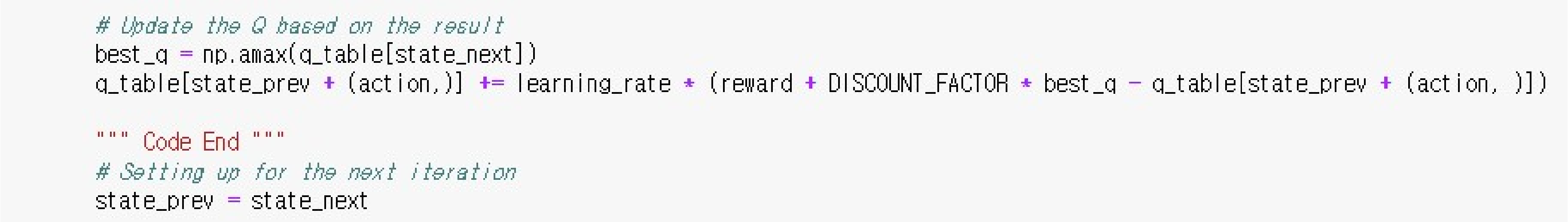


```
        # Reset the environment
        obv = env.reset()
        state_prev = observation_to_state(obv)

        for step in range(MAX_STEPS):
            """ Code Start """

            # Select an action
            action = select_action(state_prev, explore_rate)

            # Execute the action
            obv, reward, done, _ = env.step(action)

            # Observe the result
            state_next = observation_to_state(obv)
```

Simulation

At each episode...

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + \alpha_t(s_t, a_t) \cdot (R_{t+1} + \gamma \cdot max_a Q_t(s_{t+1}, a) - Q(s_t, a_t))$$



```
# Update the Q based on the result
best_q = np.amax(q_table[state_next])
q_table[state_prev + (action,)] += learning_rate * (reward + DISCOUNT_FACTOR * best_q - q_table[state_prev + (action, )])

""" Code End """
# Setting up for the next iteration
state_prev = state_next
```

Microsoft Student Partners

Microsoft