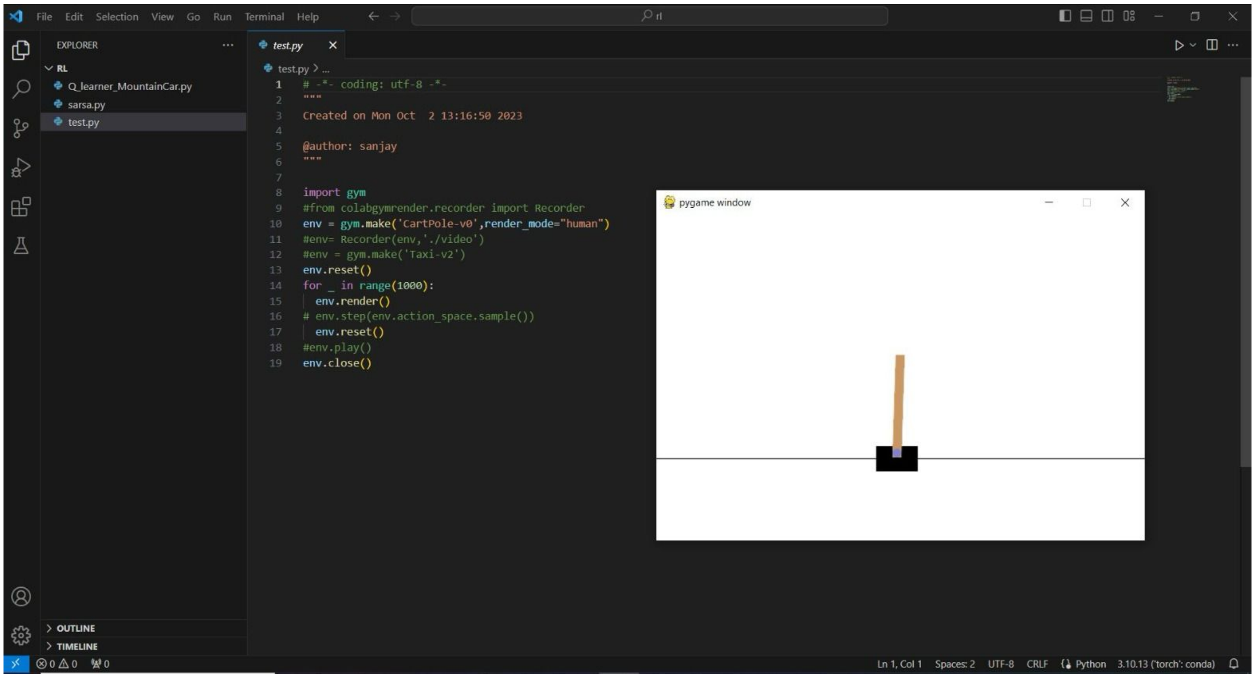
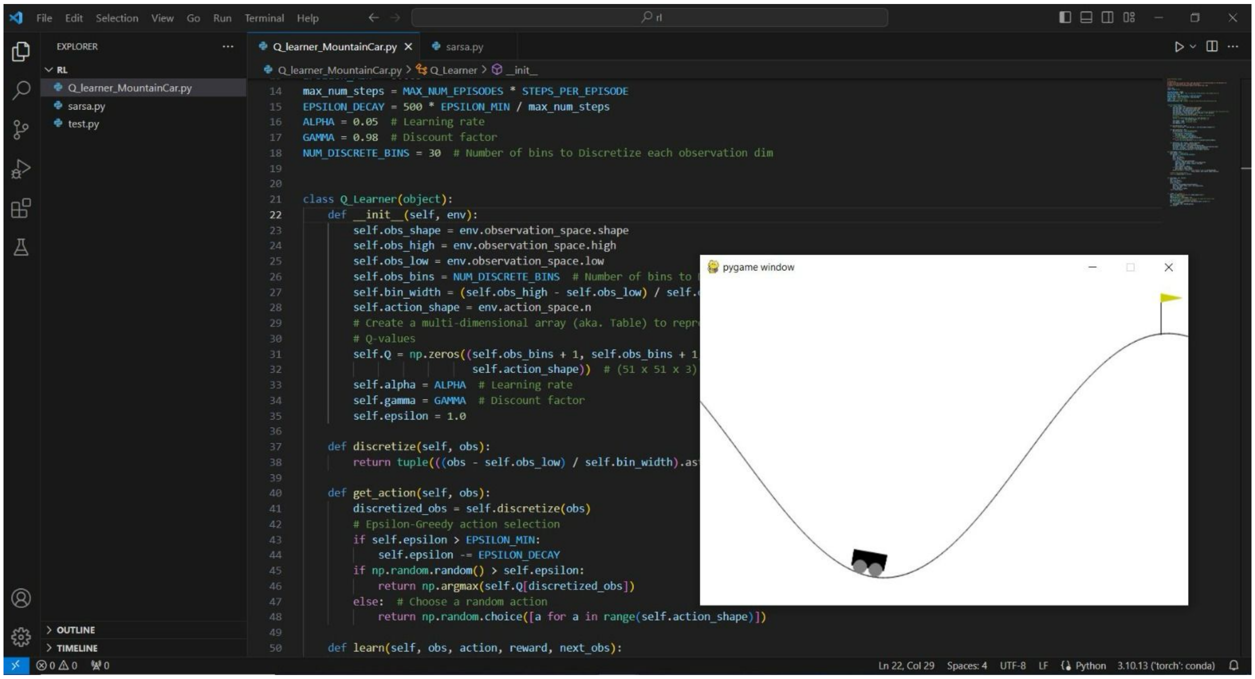
Assignment -2

OpenAI GYM:



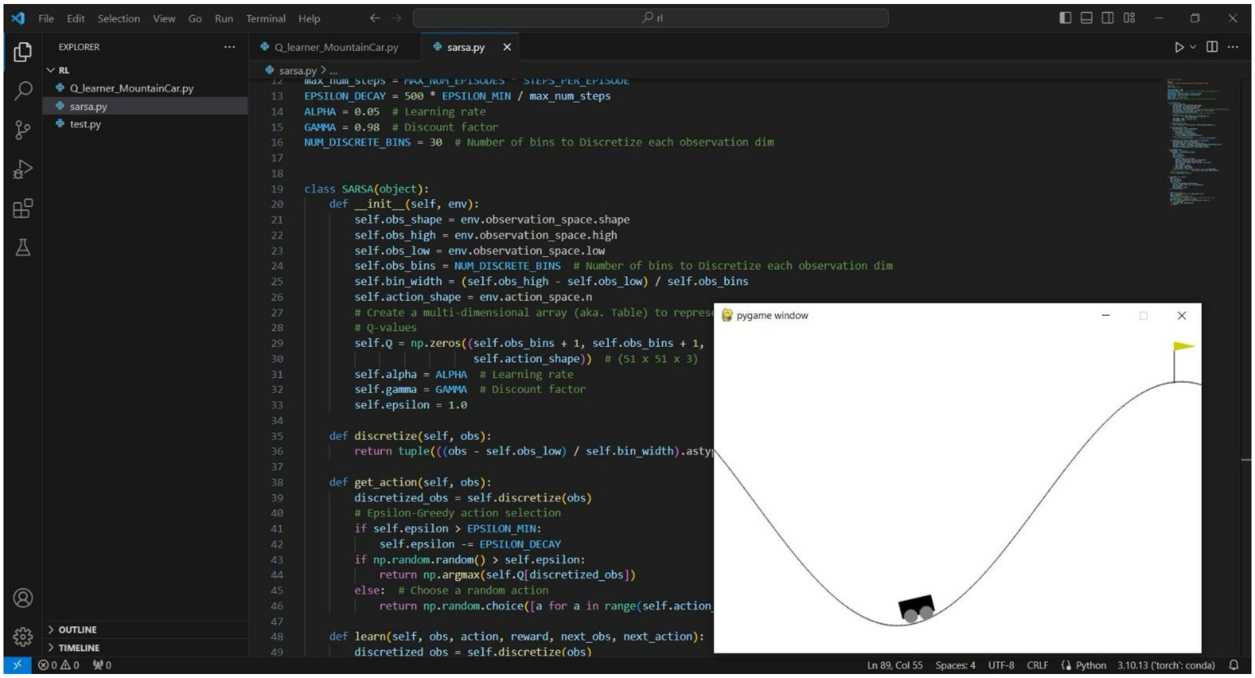
```python
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  2 13:16:50 2023

@author: sanjay
"""

import gym
#from colabgymrender.recorder import Recorder
env = gym.make('CartPole-v0',render_mode="human")
#env= Recorder(env,'./video')
#env = gym.make('Taxi-v2')
env.reset()
for _ in range(1000):
  env.render()
# env.step(env.action_space.sample())
  env.reset()
#env.play()
env.close()
```

QLearning:



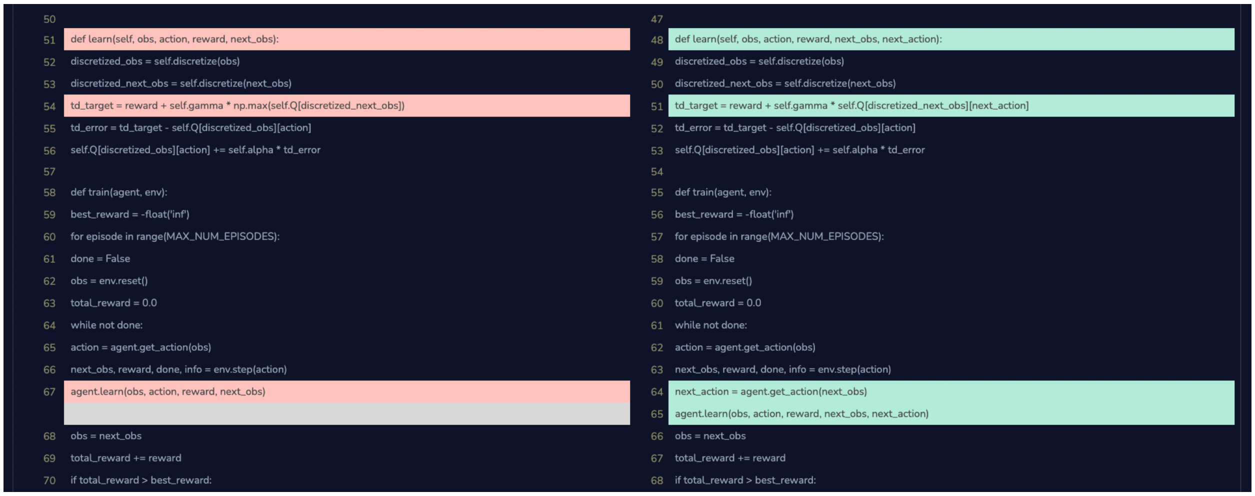
```python
max_num_steps = MAX_NUM_EPISODES * STEPS_PER_EPISODE
EPSILON_DECAY = 500 * EPSILON_MIN / max_num_steps
ALPHA = 0.05  # Learning rate
GAMMA = 0.98  # Discount factor
NUM_DISCRETE_BINS = 30  # Number of bins to Discretize each observation dim


class Q_Learner(object):
    def __init__(self, env):
        self.obs_shape = env.observation_space.shape
        self.obs_high = env.observation_space.high
        self.obs_low = env.observation_space.low
        self.obs_bins = NUM_DISCRETE_BINS  # Number of bins to
        self.bin_width = (self.obs_high - self.obs_low) / self.
        self.action_shape = env.action_space.n
        # Create a multi-dimensional array (aka. Table) to repr
        # Q-values
        self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1
                           self.action_shape))  # (51 x 51 x 3)
        self.alpha = ALPHA  # Learning rate
        self.gamma = GAMMA  # Discount factor
        self.epsilon = 1.0

    def discretize(self, obs):
        return tuple(((obs - self.obs_low) / self.bin_width).as

    def get_action(self, obs):
        discretized_obs = self.discretize(obs)
        # Epsilon-Greedy action selection
        if self.epsilon > EPSILON_MIN:
            self.epsilon -= EPSILON_DECAY
        if np.random.random() > self.epsilon:
            return np.argmax(self.Q[discretized_obs])
        else:  # Choose a random action
            return np.random.choice([a for a in range(self.action_shape)])

    def learn(self, obs, action, reward, next_obs):
```

SARSA (State Action Reward State Action):


File Edit Selection View Go Run Terminal Help
rl
EXPLORER
RL
Q_learner_MountainCar.py
sarsa.py
test.py
sarsa.py > ...
pygame window
OUTLINE
TIMELINE
Ln 89, Col 55 Spaces: 4 UTF-8 CRLF Python 3.10.13 ('torch': conda)

```python
max_num_steps = MAX_NUM_EPISODES * STEPS_PER_EPISODE
EPSILON_DECAY = 500 * EPSILON_MIN / max_num_steps
ALPHA = 0.05  # Learning rate
GAMMA = 0.98  # Discount factor
NUM_DISCRETE_BINS = 30  # Number of bins to Discretize each observation dim


class SARSA(object):
    def __init__(self, env):
        self.obs_shape = env.observation_space.shape
        self.obs_high = env.observation_space.high
        self.obs_low = env.observation_space.low
        self.obs_bins = NUM_DISCRETE_BINS  # Number of bins to Discretize each observation dim
        self.bin_width = (self.obs_high - self.obs_low) / self.obs_bins
        self.action_shape = env.action_space.n
        # Create a multi-dimensional array (aka. Table) to represe
        # Q-values
        self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1,
                           self.action_shape))  # (51 x 51 x 3)
        self.alpha = ALPHA  # Learning rate
        self.gamma = GAMMA  # Discount factor
        self.epsilon = 1.0

    def discretize(self, obs):
        return tuple(((obs - self.obs_low) / self.bin_width).asty

    def get_action(self, obs):
        discretized_obs = self.discretize(obs)
        # Epsilon-Greedy action selection
        if self.epsilon > EPSILON_MIN:
            self.epsilon -= EPSILON_DECAY
        if np.random.random() > self.epsilon:
            return np.argmax(self.Q[discretized_obs])
        else:  # Choose a random action
            return np.random.choice([a for a in range(self.action_

    def learn(self, obs, action, reward, next_obs, next_action):
        discretized_obs = self.discretize(obs)
```

Qlearning vs SARSA:



```
def learn(self, obs, action, reward, next_obs):
discretized_obs = self.discretize(obs)
discretized_next_obs = self.discretize(next_obs)
td_target = reward + self.gamma * np.max(self.Q[discretized_next_obs])
td_error = td_target - self.Q[discretized_obs][action]
self.Q[discretized_obs][action] += self.alpha * td_error

def train(agent, env):
best_reward = -float('inf')
for episode in range(MAX_NUM_EPISODES):
done = False
obs = env.reset()
total_reward = 0.0
while not done:
action = agent.get_action(obs)
next_obs, reward, done, info = env.step(action)
agent.learn(obs, action, reward, next_obs)

obs = next_obs
total_reward += reward
if total_reward > best_reward:
```

```
def learn(self, obs, action, reward, next_obs, next_action):
discretized_obs = self.discretize(obs)
discretized_next_obs = self.discretize(next_obs)
td_target = reward + self.gamma * self.Q[discretized_next_obs][next_action]
td_error = td_target - self.Q[discretized_obs][action]
self.Q[discretized_obs][action] += self.alpha * td_error

def train(agent, env):
best_reward = -float('inf')
for episode in range(MAX_NUM_EPISODES):
done = False
obs = env.reset()
total_reward = 0.0
while not done:
action = agent.get_action(obs)
next_obs, reward, done, info = env.step(action)
next_action = agent.get_action(next_obs)
agent.learn(obs, action, reward, next_obs, next_action)
obs = next_obs
total_reward += reward
if total_reward > best_reward:
```

GIT: https://github.com/sanjaybhargavm/mountain_car.git