Project Scope

Scope: This personal project involved developing a Double Deep-Q Network (DDQN) to play and master the classic game Super Mario Bros. The goal was to create an AI agent capable of learning game strategies through trial and error, maximizing in-game rewards, and completing levels efficiently. The project aimed to deepen my understanding of reinforcement learning techniques and their application in complex, dynamic environments like video games.

My Role and Solution

Role: Sole Developer and Researcher

Solution Approach: I implemented the DDQN algorithm using PyTorch, and OpenAI Gym. The solution involved building a neural network that learned to predict the best actions for Mario to take based on the current game state. By training the model over thousands of episodes, I refined its ability to overcome obstacles, defeat enemies, and complete levels with minimal retries. I also fine-tuned hyperparameters and incorporated techniques like experience replay and target networks to stabilize training.

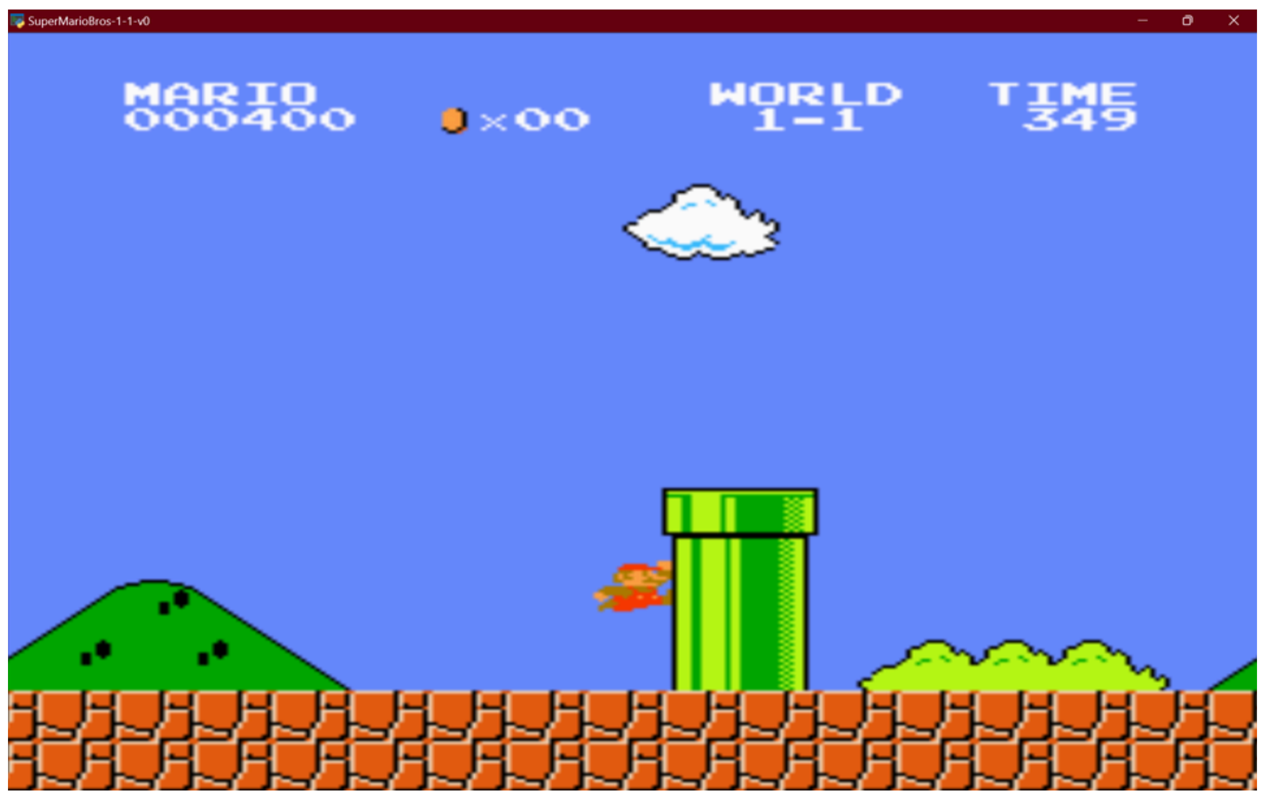
My Work Process

Involvement: I designed and coded the entire solution from scratch, starting with the integration of Super Mario Bros. into the OpenAI Gym environment. I then constructed the DDQN architecture, trained the model, and iterated on the training process by adjusting parameters and improving the reward function. This project required extensive debugging, experimentation with different strategies, and analysis of the agent's performance to achieve optimal results.

Outcome And Results Achieved

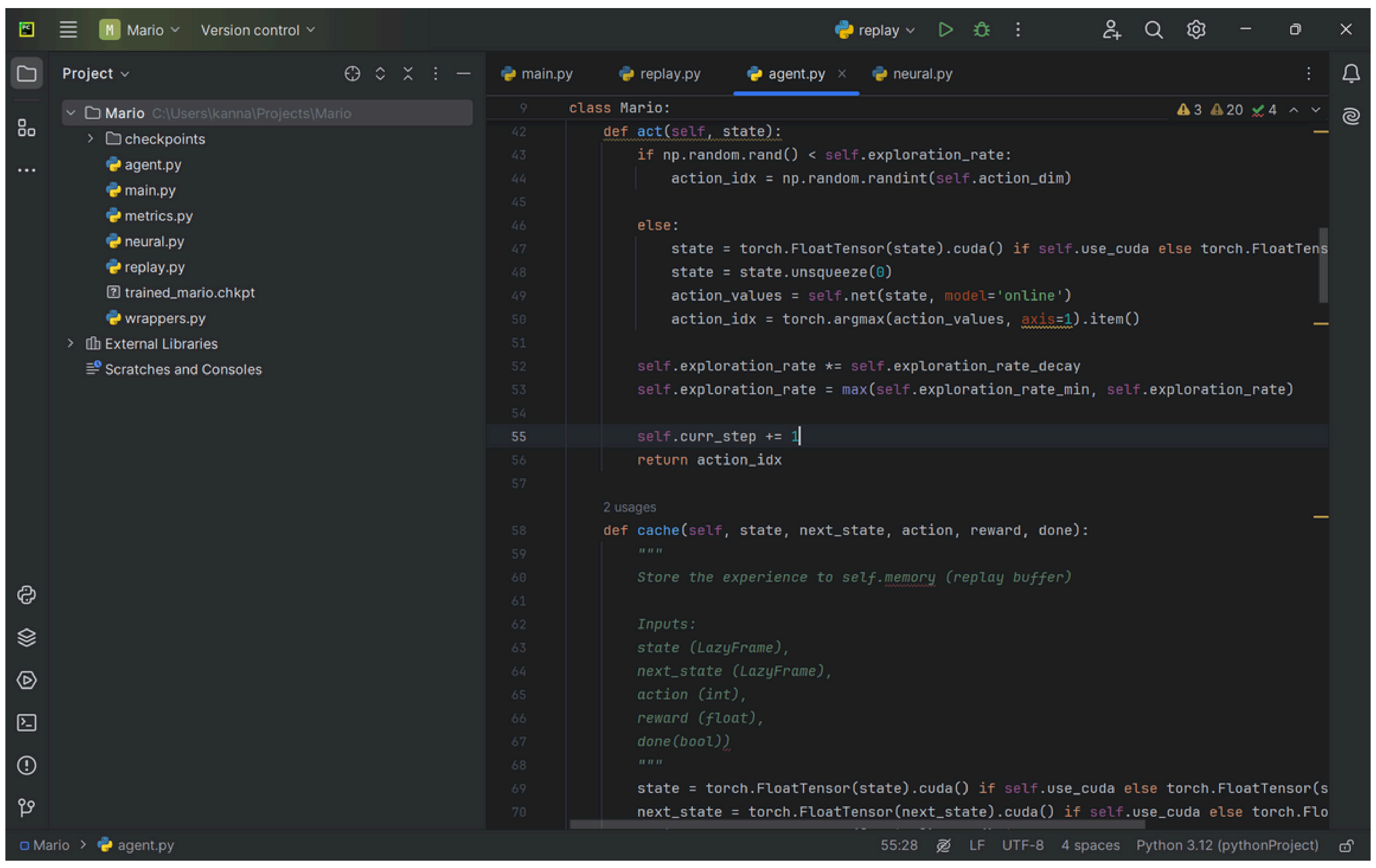
Outcome: The trained DDQN agent successfully learned to complete multiple levels of Super Mario Bros., showing a significant improvement in gameplay performance over time.

Results: The agent demonstrated advanced strategies, such as avoiding dangerous enemies and timing jumps precisely, achieving completion of levels with minimal loss of lives. This project showcased the practical application of reinforcement learning to a classic problem and further enhanced my skills in AI and deep learning.



AI in training

Excerpt of code



```python
class Mario:
    def act(self, state):
        if np.random.rand() < self.exploration_rate:
            action_idx = np.random.randint(self.action_dim)

        else:
            state = torch.FloatTensor(state).cuda() if self.use_cuda else torch.FloatTens
            state = state.unsqueeze(0)
            action_values = self.net(state, model='online')
            action_idx = torch.argmax(action_values, axis=1).item()

        self.exploration_rate *= self.exploration_rate_decay
        self.exploration_rate = max(self.exploration_rate_min, self.exploration_rate)

        self.curr_step += 1
        return action_idx

    def cache(self, state, next_state, action, reward, done):
        """
        Store the experience to self.memory (replay buffer)

        Inputs:
        state (LazyFrame),
        next_state (LazyFrame),
        action (int),
        reward (float),
        done(bool))
        """
        state = torch.FloatTensor(state).cuda() if self.use_cuda else torch.FloatTensor(s
        next_state = torch.FloatTensor(next_state).cuda() if self.use_cuda else torch.Flo
```