Q-Learning Shooter Project Report

Introduction

The Q-Learning Shooter is a machine learning-based gaming simulation designed to test the capabilities of reinforcement learning in a dynamic, game-like environment. The project leverages Q-Learning principles, specifically Double Q-Learning, to train ML agents to learn and adapt in real-time. This project demonstrates how machine learning can be applied in gaming and simulation environments for ML-controlled behaviors.

Objectives

1. To implement a Q-Learning algorithm in a gaming environment.
2. To train an AI agent to perform well in a shooter game by maximizing its rewards.
3. To evaluate the agent's ability to learn and adapt through reinforcement learning techniques.
4. To explore the challenges and limitations of Q-Learning in gaming contexts.

Past Work

This project does not build upon any prior experience or existing implementations but is a foundational attempt to apply reinforcement learning in gaming as well as my first dive into game development.

Problem Description

Creating adaptive ML behavior in games is difficult due to the dynamic and unpredictable nature of such environments. This project aims to address this by using Q-Learning, a reinforcement learning technique, to develop an ML agent that can learn and adapt to maximize performance.

- How can an AI agent aim and shoot accurately?
- How can it balance movement and strategic alignment?
- How can the agent adapt to increasingly difficult levels while avoiding penalties?

Dataset Description

Although Q-Learning does not require a pre-existing dataset, the environment acts as the dataset:

- **State Space:** Player position, enemy position, bullets, and scores.
- **Action Space:** Move left, move right, or shoot.
- **Rewards:**
    - Positive rewards for hitting the enemy (+100).
    - Penalties for missed shots, prolonged movement, and being hit (-20 to -50).
    - Level advancement adds implicit rewards as the environment increases in difficulty.

Methods Used

1. **Q-Learning:** Used to train the agent based on a reward system.
2. **Double Q-Learning:** Introduced to reduce overestimation bias by maintaining two Q-Networks.
3. **Experience Replay Buffer:** Ensures that the agent learns from diverse scenarios.
4. **Deep Learning Q-Networks (DQNs):** Neural networks that approximate Q-values for more complex environments.

Procedure

1. **Environment Design:**
    - Player moves on a 10x10 grid.
    - Enemy spawns and moves toward the bottom.

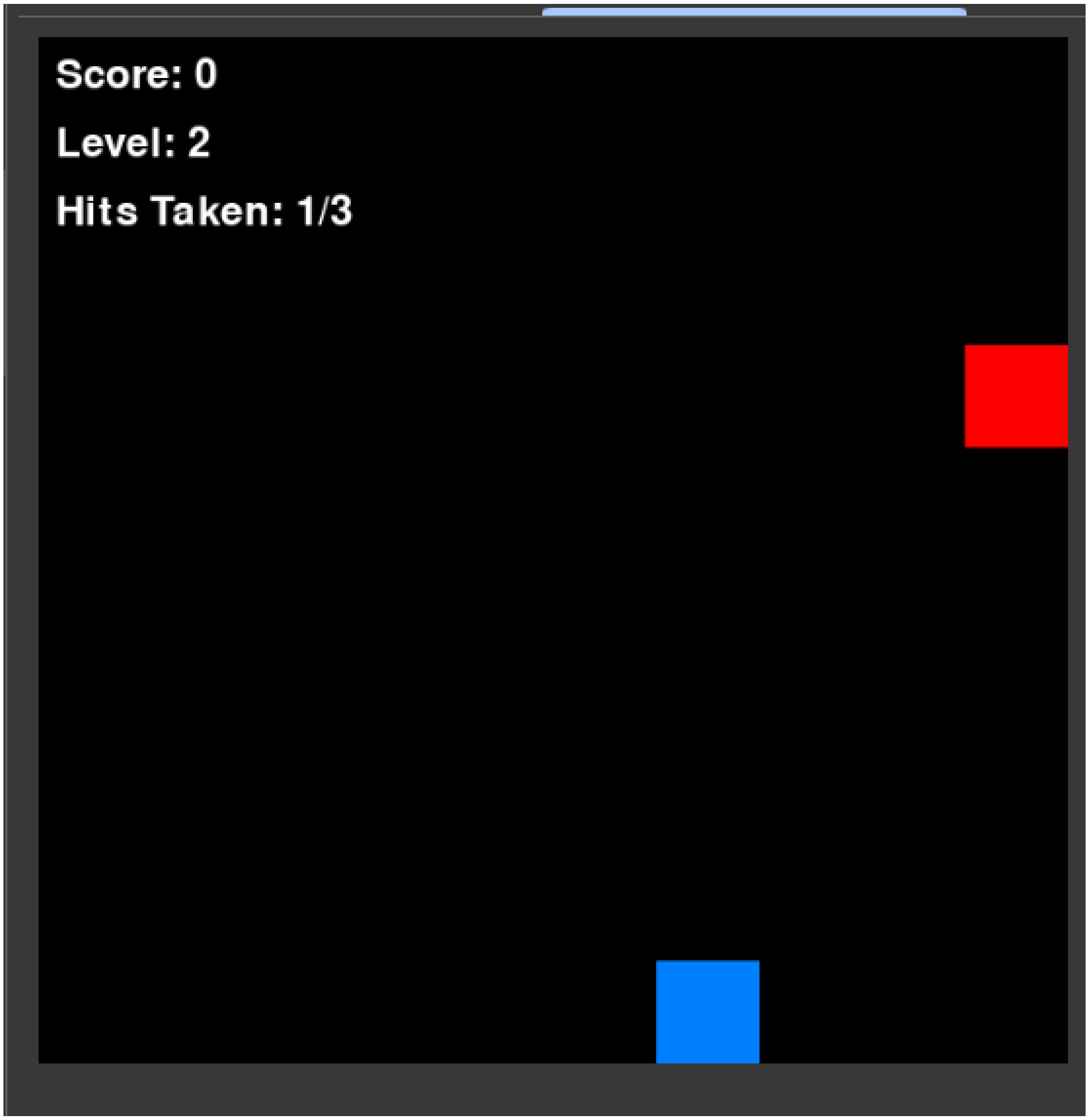
- Player fires bullets upward to hit the enemy.
- The game ends when the AI takes three hits or reaches a predefined maximum number of steps.

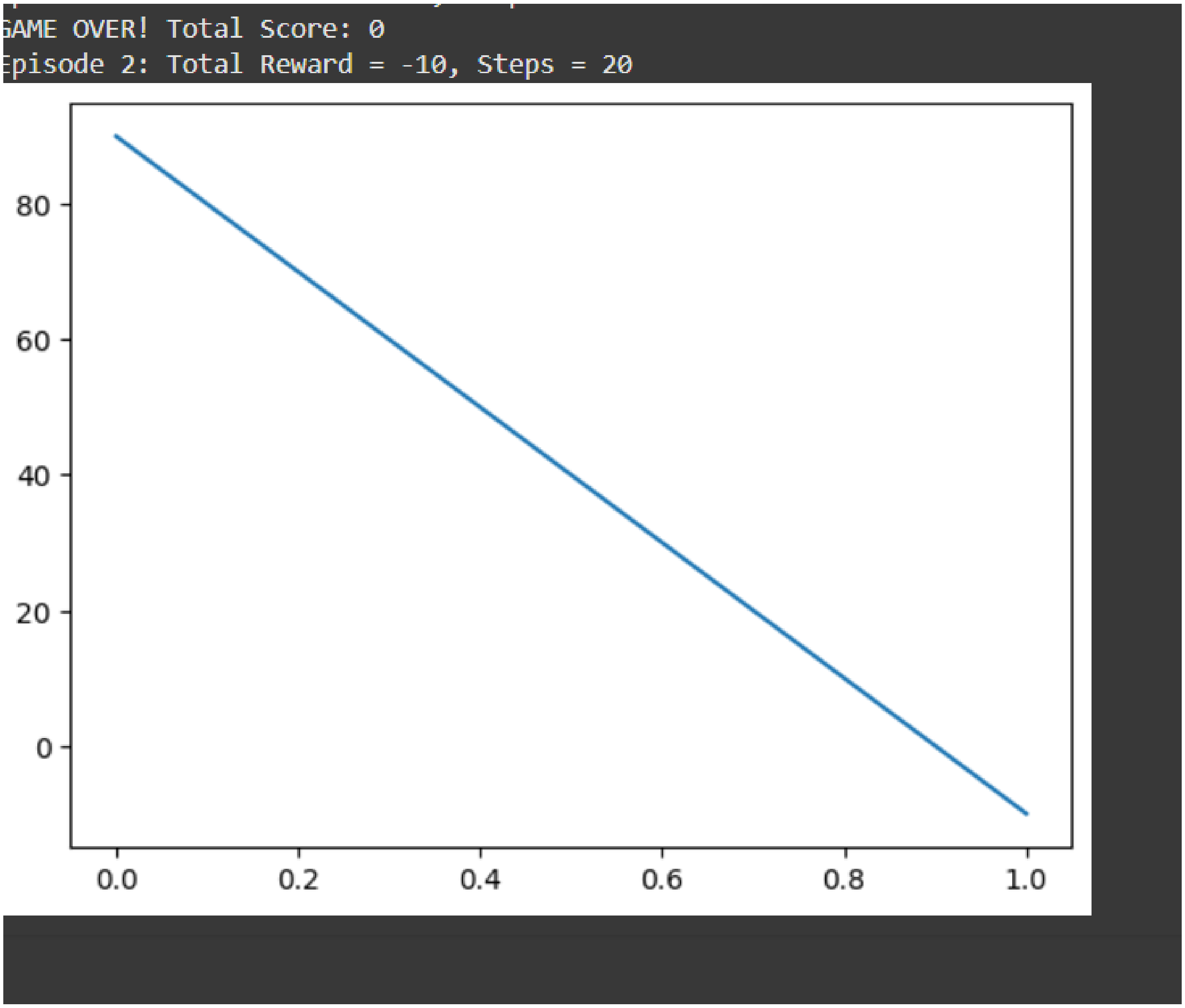
2. **Reinforcement Learning Implementation:**
   - Define the state-action-reward system.
   - Initialize Q-Tables and two neural networks for Double Q-Learning.
   - Train the agent across 500 episodes, with rewards and penalties guiding its learning.
3. **Hyperparameter Tuning:**
   - Learning rate (α), discount factor (γ), exploration-exploitation tradeoff (ε), and reward structure
4. **Game Visualization:**
   - Results and performance visualized through gameplay and score tracking.
   - A GIF is generated to showcase the agent's progress.

Results

1. **Gameplay Performance:**
   - The AI successfully advances levels and demonstrates learning behavior, though progress varies with difficulty.
   - Rewards per episode reflect the agent's learning curve.
2. **Visuals:**
   - The generated GIF showcases gameplay, including score tracking and AI behaviors.

Screenshots:





Limitations

1. **Memory Limitations:**
    - The replay buffer and GIF generation require significant memory, leading to potential performance issues.
    - Computational resources can limit the size of the environment or the complexity of the agent.
2. **Learning Inefficiency:**
    - Q-Learning is slow to converge in environments with large state-action spaces.

    - The agent struggles with longer-term planning and balancing exploration vs. exploitation.

3. **Simplified Game Environment:**
    - The grid-based shooter is simplistic, and performance may not scale in more complex, real-world games.

Conclusions

The Q-Learning Shooter project demonstrates the feasibility of reinforcement learning in gaming environments. By implementing Double Q-Learning and tuning reward structures, the AI agent can exhibit intelligent behavior and adapt to dynamic challenges. However, the limitations highlight areas for future work, such as:

- Incorporating more advanced RL techniques (e.g., Proximal Policy Optimization or Actor-Critic methods).
- Scaling the environment to include more complex game mechanics.
- Optimizing memory usage for real-time and large-scale applications.
- Incorporate 3D Sprites
- First or Third Person perspective using Pandas3D or Unity Game Engine
- Reward Structure
- Game Loop