Pokémon Battle Arena: An Experiment in Reinforcement Learning

Team ID: CU_CP_Team_10196

Team Leader: Jayal Shah
Team Member 1: Mayank Jangid
Team Member 2: Dhairya Jadav

Mentor: Palwinder Singh

1. Abstract

This report details the design, implementation, and evaluation of an autonomous AI agent trained to master a strategic Pokémon battle. The central problem was to determine if a "blank slate" agent, with no pre-programmed rules, could learn effective combat strategies purely through trial-and-error. To solve this, we custom-built a PokemonBattleEnv environment using the Python gymnasium library, modeling HP, energy, and a type-advantage system. An agent was then trained using the Proximal Policy Optimization (PPO) algorithm from stable-baselines3 for 100,000 timesteps. The trained agent was evaluated in a tournament against opponents of varying types. The results demonstrate that the agent successfully learned foundational strategies, such as winning neutral-advantage matchups and identifying unwinnable fights. However, the experiment also revealed emergent, non-optimal behaviors (e.g., stalling in an advantageous matchup), highlighting the critical sensitivity of agent behavior to its "reward signal." This project successfully provides a robust foundation for future research into more complex AI for strategy games.

2. Introduction

2.1. Project Motivation

In recent years, Reinforcement Learning (RL) has emerged as a dominant paradigm in Artificial Intelligence, with agents achieving superhuman performance in complex games like Go, Chess, and StarCraft. These successes raise a compelling question: how far can this "learning by doing" approach be pushed? This project was motivated by a desire to apply these advanced techniques to a beloved and strategically deep domain: Pokémon battles.

2.2. Problem Statement

The core question of this project is:
"Can an AI agent, given no pre-programmed rules, learn to master a complex strategy game?"
Unlike a simple game, a Pokémon battle involves managing resources (HP, energy), anticipating opponent actions, and exploiting hidden information (type advantages). Our goal was to build an agent that could learn a winning strategy from the ground up.

2.3. Core Challenges

This problem presented three primary technical challenges:

1. **The "Blank Slate" Problem:** The agent begins with zero knowledge. It does not know that "fire is good against grass" or that "running out of HP is bad." The system must be designed to allow it to learn these concepts purely through interaction.
2. **The Environment Challenge:** No off-the-shelf Pokémon simulator was suitable for this task. We had to build a custom gymnasium environment from scratch that accurately modeled all game logic, including HP, energy costs, and type-advantage multipliers.
3. **The "How to Win" Problem:** How do we define "winning" for an AI? We needed to design a precise "Reward Signal" that would guide the agent toward effective fighting, balancing aggression with defensive play and resource management.

2.4. Project Objectives

To address these challenges, we set the following objectives:

1. **Build:** Develop a stable, custom Python environment (PokemonBattleEnv) that simulates a 1v1, turn-based battle.
2. **Train:** Implement a Proximal Policy Optimization (PPO) agent and train it over thousands of simulated battles against random opponents.
3. **Evaluate:** Deploy the trained agent in a "tournament" against a fixed set of diverse opponents to measure its performance.
4. **Analyze:** Automatically generate battle videos to visually analyze the agent's learned strategies and emergent behaviors.

3. Methodology and System Architecture

3.1. Core Technologies

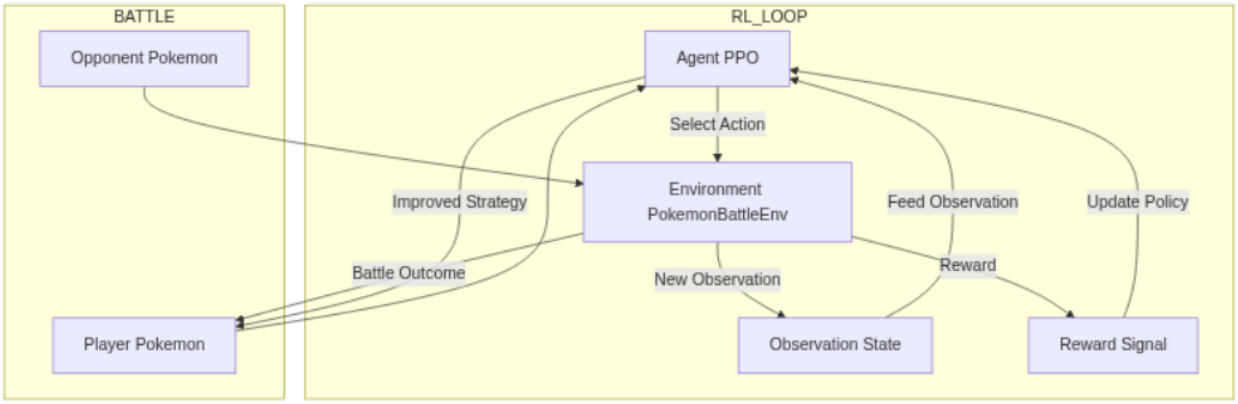
This project was built entirely in Python, leveraging several key open-source libraries:

- **gymnasium:** The modern fork of OpenAI Gym, used as the foundational API for creating our custom PokemonBattleEnv environment.
- **stable-baselines3:** A leading library for RL algorithms, providing a robust and GPU-compatible implementation of PPO.
- **torch:** The deep learning framework used by stable-baselines3 to power the agent's neural network.
- **moviepy & opencv-python:** Used to dynamically create the "cinematic" MP4 video recordings of

the agent's battles.
- **pillow (PIL):** Used for fetching and compositing Pokémon sprites onto the video frames.
- **requests:** Used to fetch Pokémon sprite data from the PokeAPI.

3.2. System Architecture: The RL Loop

The project is built on the standard agent-environment loop, which functions as the "brain" of the system.



The flow is as follows:

1. **Observation:** The **Agent (PPO)** observes the current Observation State from the **Environment**.
2. **Action:** Based on this state, the agent's policy selects an Action (e.g., "Light Attack").
3. **Step:** The Environment processes this action, simulates the opponent's move, and calculates the results.
4. **Feedback:** The Environment returns two things to the agent:
   - A **New Observation** (the updated game state).
   - A **Reward Signal** (a number indicating if the action was "good" or "bad").
5. **Learn:** The agent receives this feedback and uses the PPO algorithm to Update Policy, making it slightly more likely to take "good" actions in the future. This loop is repeated 100,000 times.

3.3. The Agent: Proximal Policy Optimization (PPO)

We chose the Proximal Policy Optimization (PPO) algorithm as our agent. PPO is a state-of-the-art choice because it strikes a balance between sample efficiency (learning quickly) and stability. It is an "on-policy" actor-critic algorithm, meaning it learns both a "policy" (the "actor," which decides what to do) and a "value function" (the "critic," which estimates how good the current situation is).

4. Custom Environment Design: PokemonBattleEnv

The core of this project was the custom environment. We designed it to be a simplified, yet strategically valid, 1v1 battle.

4.1. Core Game Logic

We implemented the following rules, as seen in the .ipynb file:

- **Stats:** Each Pokémon starts with 100 HP and 100 Energy.
- **Types:** We implemented 5 core types: Fire, Water, Grass, Electric, and Normal.
- **Type Advantage:** A logic map dictates combat multipliers.
  - **Super Effective (1.5x damage):** Fire > Grass, Water > Fire, Grass > Water, Electric > Water.
  - **Not Very Effective (0.5x damage):** Fire < Water, Water < Grass, Grass < Fire.
  - **Neutral (1.0x damage):** All other combinations.

4.2. Observation Space (The "Senses")

The agent cannot "see" the game. It only receives an array of 4 numbers (a Box space) representing the complete game state:

- [Agent HP, Agent Energy, Opponent HP, Turn Number]
  This simple state representation is a limitation, as the agent does not know its own type or the opponent's type. It must infer these properties by observing the damage it deals and takes.

4.3. Action Space (The "Moves")

The agent has 4 possible moves to choose from at each turn (a Discrete(4) space):

- **Action 0: Light Attack:** Low damage (10-20), low energy cost.
- **Action 1: Heavy Attack:** High damage (25-40), high energy cost.
- **Action 2: Guard:** Reduces incoming damage by 50% for one turn.
- **Action 3: Recover:** Heals a random amount of HP (10-25).

4.4. The Reward Signal (The "Motivation")

This is the most critical component, as it is the *only* thing that teaches the agent. Based on the .ipynb code, we designed the following reward signal:

- **Reward for Damage:** The agent is rewarded for dealing damage.
  - Light Attack: reward += 0.5 * damage_dealt
  - Heavy Attack: reward += 0.8 * damage_dealt (This encourages using stronger moves).
- **Penalty for Living:** The agent receives a small penalty for every turn that passes.
  - Per Turn: reward -= 0.5
    This penalty discourages the agent from stalling or using "Guard" and "Recover" indefinitely, forcing it to find a way to end the battle.

5. Training and Evaluation

5.1. Training Process

The agent was trained on Google Colab using a GPU.

- **Algorithm:** PPO
- **Policy:** MlpPolicy (a multi-layer perceptron neural network)
- Total Timesteps: 100,000
  During training, the agent's USER_POKEMON_ID was set to 6 (Charizard, a Fire type), and it battled against a new, randomly selected Pokémon (ID 1-151) for each "episode" (battle).

5.2. Key Feature: Cinematic Battle Visualization

A key "wow factor" of this project is the automated evaluation system. After training, the agent battles its tournament opponents, and the system automatically:

1. Captures each "frame" of the battle.
2. Uses Pillow to draw the sprites, HP bars, and action text onto the frame.
3. Uses OpenCV and MoviePy to stitch these frames into a cinematic .mp4 video.
4. Adds custom intro and outro cards to the video, declaring the matchup and the winner.

5.3. Tournament Results and Analysis

We evaluated the trained agent (Pokémon #6, Charizard [Fire]) against 5 distinct opponents. The results from the notebook and presentation are as follows:

Agent	Opponent	Opponent Type	Advantage	Result	Battle Log Analysis
Charizard (Fire)	#26 Raichu	Electric	Neutral	**WIN**	A balanced fight. Agent won in 13 turns.
Charizard (Fire)	#66 Machamp	Normal	Neutral	**WIN**	A long battle of attrition. Agent won in 28 turns.

Charizard (Fire)	#37 Vulpix	Fire	Neutral	**WIN**	An aggressive, quick victory in 6 turns.
Charizard (Fire)	#9 Blastoise	Water	**Disadvantage**	**LOSS**	Agent was quickly defeated in 4 turns.
Charizard (Fire)	#12 Butterfree	Grass	**ADVANTAGE**	**LOSS**	Agent lost on 50-turn timeout.

Analysis of Emergent Behavior:

The tournament data reveals a fascinating and successful learning curve, with one critical exception:

- **Learned Strategy (Success):** The agent correctly learned to be aggressive in neutral matchups (vs. Electric, Normal, Fire), leading to consistent wins. It also learned that the Water matchup was unwinnable, losing quickly. This proves the "blank slate" agent successfully learned basic strategy.
- **Emergent Flaw (The "Grass" Problem):** The most interesting result is the **loss to Pokémon #12 (Grass)**. As a Fire type, the agent had a 1.5x damage advantage and should have won easily.
    - **Why did it lose?** The battle log from the .ipynb shows the agent spammed "Recover" for nearly 50 turns, dealing very little damage. It lost because the battle timed out.
    - **The Cause:** This is a classic RL problem. Our reward signal (reward -= 0.5 per turn) taught the agent that "surviving" was good, but it didn't *sufficiently* reward the *extra* damage from its 1.5x advantage. The agent found a "local optimum" where it could heal itself and avoid HP loss, but it failed to learn the "global optimum" of ending the fight quickly.

6. Conclusion

This project was a successful end-to-end implementation of a Reinforcement Learning system. We successfully built a custom gymnasium environment from scratch, proving that a "blank slate" PPO agent can learn foundational game strategies without any pre-programmed rules.

The agent demonstrated that it could:

1. Win neutral matchups through effective resource management.
2. Identify and lose quickly in unwinnable, type-disadvantaged matchups.

The experiment's most valuable insight came from its failure: the agent's inability to defeat a type-advantaged Grass opponent. This demonstrates the high sensitivity of RL agents to their reward signal and highlights that simply "not losing" is different from "learning to win." This project provides a

strong, flexible foundation for building much more complex AI agents in the future.

7. Future Work

Based on our analysis, we can build on this foundation by making the agent "smarter" in three key areas:

1. Smarter Rewards (Reward Shaping):
   The "Grass" problem can be fixed by "shaping" the reward. We would add a large, one-time reward for winning (+100) and a large penalty for losing (-100). This would explicitly teach the agent that winning is the true objective, not just surviving.
2. Smarter Sight (Richer Observations):
   We can make the agent "smarter" by giving it better "eyes." We would add the agent's own type and the opponent's type directly to the observation space. This would allow it to learn advanced, explicit strategies like, "IF opponent is Grass, always use Heavy Attack."
3. Smarter Actions (Deeper Mechanics):
   The environment can be expanded to include more complex game mechanics, such as status moves (e.g., "Paralyze," "Sleep") or scaling the project to a full 6v6 Pokémon battle, which would be a significant and novel research challenge.