A Performant Game Engine and Monte-Carlo Tree Search Agent for Pokemon Battles

Derek Dong, Computer Science, Yale University
Advised by James Glenn, Computer Science, Yale University

OVERVIEW

The design and building of agents that play video games is a key problem in AI/ML.
> Games have controlled environments where techniques can be refined and then transferred to more practical applications.

Pokemon is a popular role-playing strategy video game.
> 25+ million units were sold of the most recent main game, Pokemon Scarlet/Violet.
> At the core of a Pokemon game is a stochastic turn-based game that is very complex.

Fast (and, thus, cheap) simulation of state transitions is important to training stronger agents.
> Modern agents tend to be taught how the game works by *lots* of examples.
> Recorded match histories between human players are limited, so generating the data via *simulated* battles is crucial.

> Our goal: to provide ML researchers with a ready-made, *efficient* Pokemon simulation engine to serve as the backend for their Pokemon-playing agents.

CONTRIBUTIONS

> Built a Pokemon Gen-8 single-battle engine in C++ that runs turns ~**100 times faster** than the standard, Pokemon Showdown.

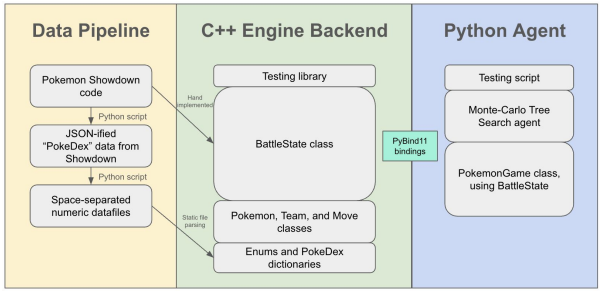
> As an example for using the engine, wrote a Monte Carlo Tree Search agent with a **90+% win-rate** in the tested battles.

DESIGN CHOICES

Advantages of C++ over TypeScript:
> Has more rigid typing
> Can maximize stack-allocation
> Has well-tested libraries for interfacing from Python

Additional benefits of the engine over Pokemon Showdown:
> Is a stateless-server model, meaning better testing and the ability to re-sample state-action steps as much as desired
> Has a more rigid control flow, meaning it's easier to read the code and understand the mechanics of the game.

SYSTEM DESIGN



RESULTS (ENGINE)

Our engine was *2-4 orders of magnitude* faster than Pokemon Showdown. For the most common types of turns, our engine was **100-200 times faster**. Note:
> We had to time Pokemon Showdown internally, i.e. by Javascript `console.time()` and `console.timeEnd()` calls.
> We also measured our engine's performance from within our C++ testing suite.

Experiment	System	Trials	Mean Time (μs)	Std. Dev. (μs)
Start battle	Pokemon Showdown	20	1914	17.218
	Engine	1000	0.307	0.000910
Run turn	Pokemon Showdown	20	2768	23.922
	Engine	1000	17.542	4.56
Switch	Pokemon Showdown	20	461	63.318
	Engine	1000	1.964	0.228

EXAMPLE

```
import sys
import os
# Add the compiled module path to sys.path
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "../build/
bin")))
from fast_pkmn import *
state = BattleState()
teams = [Team(), Team()]
teams[0].add_pokemon(0, get_charmander(lvl=5))
teams[1].add_pokemon(0, get_bulbasaur(lvl=5))
state.set_team(0, teams[0]), state.set_team(1, teams[1])
state.start_battle()
state.teams[0].choices = Choice(False, 1, -1)
state.teams[1].choices = Choice(False, 0, -1)
state = state.run_turn()
```

GITHUB

RESULTS (AGENT)

We wrote a simple MCTS agent with the following win rates against the real Pokemon Run and Bun opponent logic:

Table 2: Data from running the MCTS agent with varying teams and against various opposing teams: averages over 500 games with 0.1 seconds of decision time. Win rates, the average number of rollouts achieved per-turn, the average number of turns in a game, and the average non-fainted Pokemon conditional on a win.

Team Name	Opponent	Win rate	Rollouts	Turns	Left
Piplup8	Calvin	14%	70	6.48	1
Piplup10	Calvin	91%	62	7.72	1
Piplup8+Pooch8	Calvin	36%	57	11.69	1.15
NoItem	Aqua Grunt	22%	138	13.18	1.95
Berries	Aqua Grunt	58%	56	11.97	2.01
Berry+MaxIV	Aqua Grunt	81%	52	13.15	2.58

DISCUSSION

It's still unclear exactly how much more cost-effectively an agent can be trained and/or run with the performance gains.
However, we expect a direct comparison of agents built with Showdown and our engine as the backend will even more strongly favor our engine, because:
> Our measurements don't include I/O time
  > Our engine uses PyBind to directly create a compiled shared-object file for import, while
  > Showdown's sockets have transport-protocol overhead.
> The simulator's speed matters more if model inference is fast.
  > Less time proportionately is spent computing or waiting for a model.
  > In [Wan24], simulation was the bottleneck at inference.

FUTURE WORK

Our engine computes Pokemon battle state transitions much faster than the existing standard. For the engine to be useful in, for example, hardcore Nuzlocke runs of Run and Bun, we must implement more idiosyncratic mechanics. This includes:
> Idiosyncratic moves, abilities, etc.
  > Examples: move-copying (e.g. by Instruct) and last-damage multipliers (e.g. by Bide)
> Idiosyncratic opponent AI.
  > Examples: self-destructing moves (e.g. Explode) and setup moves (e.g. Stealth Rock)

REFERENCES

[Wan24] Jett Wang. Winning at Pokemon random battles using reinforcement learning. Master's thesis, MIT, 2024.