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

Derek Dong, James Glenn (advisor)

March 6, 2025

Abstract

An important barrier to development of Pokemon battling agents with machine learning is the limited quantity of game history data. We build an engine with state-of-the-art speed in Pokemon battle simulation to aid in developing deep/reinforcement learning agents for Pokemon battling. We demonstrate the engine's performance by comparing two Monte Carlo Tree Search agents – one that simulates battles using the engine and another that simulates battles with the closest competitor, Pokemon Showdown. We find that [INSERT HERE]

1 Introduction

Pokemon is one of, if not the, most popular and highest-grossing media franchises in the world [pkm]. Its foundation lies in role-playing games centered around a complex "turn-based" battle system where players pit the team of Pokemon they've caught against opposing teams. Players can battle against each other or against rule-based opponents built into the games. The largest Pokemon tournament each year has a prize pool of over $2,000,000, and YouTube videos detailing playthroughs of the base games or their "ROM hacks" can each garner millions of views.

Developing agents to play video games is also of particular interest in the ML community. Agents have been successfully trained to excel and even surpass human performance in games ranging from reasonably complex combinatorial games like chess to higher-branching dynamic, stochastic, and/or multi-agent games like Dota, Starcraft, and Quake [RP20]. Games have controlled, well-defined environments and, simultaneously, sufficient complexity where techniques can be demonstrated or refined and then transferred to more practical applications of high-agency AI, like robotics or large language models.

Combining the two, a research problem of interest to both Pokemon's vast playerbase and academic researchers is to develop an AI skilled at Pokemon battling. Many modern approaches to developing video game agents involve state representations and vast repositories of game histories to indirectly encode mechanics. That is, agents tend to be trained to learn how the game works by lots of examples instead of inherently-coded rules. There are limitations recording match histories between human players such as battle animation runtime, competitive format restrictions/regulations, and lack of infrastructure in all but the biggest tournaments, so generating the data via simulated battles is crucial.

The goal of the project is to provide *efficient* Pokemon battle simulation in order to generate ample game history data and enable training powerful battling agents. In this paper, use the terms "performance" and "efficiency" with respect to the objective, i.e. speed. We aim to provide for ML researchers with a well-understood environment containing a high stochastic branching factor, and we aim to democratize access to high-quality Pokemon battles (e.g. between two agents or between a player and an agent) and bring more serious Pokemon battling to the public eye.

The source code and documentation for the project can be found at `https://github.com/derekdong623/CPSC490`.

2 Background

2.1 Pokemon Battles

The basic process of a Pokemon battle consists of two players ("trainers") with "teams" of up to 6 Pokemon each. Each Pokemon has its own stats, moves, and HP values. At any point in the match, each player has one Pokemon active, and they simultaneously can choose in a turn to attack with one of its moves or switch to one of their 5 other Pokemon, for a branching factor of about 9. Once a Pokemon's HP reaches 0, they "faint," and when a trainer's Pokemon all faint, they lose.

The computational difficulty of battling comes from the complex interactions and massive state space. There are over 1,000 distinct Pokemon, not including form changes. Each Pokemon has up to two of 18 types, has one of over 200 Abilities, might be holding one of over 300 Held Items (given to it before the battle by its trainer), and has access to up to 4 of over 700 moves (taught to it before the battle by its trainer). There are further restrictions on which Ability or moves a type of Pokemon can have. Stat changes, multi-hit moves, move priority, and Statuses (e.g. Sleep or Paralysis) expand the state space, as do combinations of environmental conditions like "weather," "terrain," and "hazards." An agent must understand all of these conditions/effects and how they interact – both with the rest of their team and the opposing team's – to win the game.

A player can battle against a rule-based opponent hard-coded into the game or against another player. So there are three types of participants: a human player, an artificial agent (the development of which is the goal of the field), and the rule-based opponent. We distinguish the latter two as "AI" and "opponent" to avoid confusion.

2.2 Related Work

Pokemon battles, in their original form, are in a video game and meant to be played by a human. This, combined with the fact that the games have strict anti-piracy measures, means that they are inseparable from the GUI and animations. This is rarely a problem for human players, but a severe performance and interface bottleneck. An AI can only interface with the base games by simulating clicks and parsing/processing entire still frames to understand the game state. Most techniques also require significant data, and therefore the speed for a large number of playthroughs. Thus, battle simulators are a crucial tool for research in the field.

One popular existing framework to simulate Pokemon battles is Pokemon Showdown [sho, LT17]. The implementation allows players to easily create their own teams (instead of catching/training them from scratch in the base games) and battle each other in their browsers. It also provides a protocol for programs to send input, circumventing the GUI/animation requirements. However, it's built for generality and security. Playing against an opponent that chooses moves that are strong but impossible for their Pokemon to have (recall that such pairs maybe restricted, especially in certain formats) wouldn't be fun to play against, and so Pokemon Showdown performs certain validations on each input. Showdown also supports features like a timer (to prevent players from refusing to choose a move) and move changes (so long as the other player hasn't chosen their move yet) for UX that present overhead in both the underlying algorithm and the design choices of the system itself, being implemented in TypeScript for dynamism.

Additionally, interesting work has been done in training agents themselves. The two most common approaches are MCTS [Lin19] and reinforcement learning [HL19, SRLR20], and the most impressive work (from 2024) uses a neural network with MCTS similarly to AlphaGo to

achieve a peak of top-8 rank in the Pokemon Showdown ladder for its event [Wan24]. These works tend to overcome performance issues by aggressively parallelizing training/simulation, but a higher-performance engine should allow more work to be concentrated on developing the approach itself.

2.3 Contribution

The primary contribution of this project is to provide a performant environment/engine for further development of artificial agents, and in particular deep/reinforcement learning agents, for Pokemon battling. We reference the Pokemon Showdown simulator for exact mechanics (e.g. priorities or order of operations), but we remove legality checks and simplify the battle stream interface to reduce performance overhead. We also implement a basic Monte-Carlo Tree Search algorithm to showcase the engine, and benchmark its performance against the opponent in a few pre-determined match-ups.

We specifically implement the mechanics and opponent from Pokemon Run and Bun, a ROM hack of the Pokemon Emerald game. We choose this game because of its popularity, relative modernity of mechanics, and most importantly, clearly documented opponent behavior. However, a goal of the project is to produce a code-base that can be adapted to other games' mechanics with minimal pain.

3 Implementation

3.1 TypeScript vs. C++

The goal is to speed up simulation compared to Pokemon Showdown. The most important source of speed comes from simply changing languages from TypeScript to C++. We explain a few reasons for this: generally, TypeScript emphasizes portability while C++ emphasizes performance.

TypeScript has a far more flexible typing system. There's only one numerical type (which could be e.g. an integer or a floating point number), functions can be assigned to variables and passed around, null/undefined values are inherently supported for types and object properties, and it's possible to define a type as allowing any one of several unrelated types.

The developers of Pokemon Showdown use this to implement the effects of mechanics (e.g. moves) via a huge array of callbacks that can be defined per-mechanic in neatly organized dictionaries. Then, in each turn, actions are added to a priority queue as callbacks, popped one by one, and executed. These callbacks can, in turn, add more callbacks to the queue to be executed and so on. The level of meta-programming means a new contributor doesn't need to understand every mechanic that has already been implemented if they want to add a new mechanic, but also makes it harder to interpret how the system is functioning as a whole and exposes the system to very subtle bugs.

Data storage is also different between TypeScript and C++. With TypeScript, all objects are stored with dynamic allocation on the heap and properties for each object are stored in a hash-map (also located on the heap), requiring garbage collection to run continuously, but allowing succinct expression that Pokemon Showdown developers make generous use of. C++ objects can be statically allocated on the stack if dynamic memory usage isn't needed, and otherwise generally require manual memory management. We therefore strive to use static allocation wherever possible.

To summarize, TypeScript's flexibility allows fast development speed, effective adaptation to new Pokemon generations, and centralized storage of data (Pokemon, item, format, etc. mechanics), on top of Javascript's easy integration with front-end web-app development, all helpful to the developers of Pokemon Showdown. C++ has more rigid rules to accommodate

compilers which provide the raw performance we are looking for, and the rigidity has the added benefit of making the internal battling logic more transparent.

3.2 A (Brief) Overview of Pokemon Mechanics

We illustrate the complexity of the project by reviewing some of the mechanics we implement. First, Pokemon Run and Bun is implemented with Generation 8 (i.e. Sword/Shield and Legends: Arceus) Pokemon and mechanics. There are 905 species of Pokemon (and 170 additional "Forme" variations of some of them). Of these 1075 entities, 409 are unobtainable by the player, and 182 are further never on an opponent's team, many with idiosyncratic abilities. So even though the choice of game to implement reduces the scope of the project, it isn't by much.

Generations 6 through 9 introduced Mega Evolution, Z-Moves, Dynamax/Gigantamax, and Terastallization, respectively. These are all supported in Pokemon Showdown even though no two mechanics have ever simultaneously been present in the same generation for legacy purposes, but we only implement Mega Evolution as it's the only mechanic of these that Run and Bun includes.

At its core, a Pokemon's move attacks a defending Pokemon and deals damage based on the move's "Base Power" and the attacker's and defender's stats. There are two categories of damage, "Physical" and "Special" – a Physical move only takes the "Attack" and "Defense" stats into account, whereas a Special move only considers the "Special Attack" and "Special Defense" stats. Some moves can also apply "stat modifiers" to a Pokemon for as long as the Pokemon is on the field. Each stat can have an integer modifier in $[-6, +6]$, multiplying the stat by as low as $1/4$ or as much as 4 in various calculations. The move's Base Power can also be multiplied by other moves in that turn, abilities, or items, and the final damage value can further be multiplied by abilities, items, or environmental effects. Finally, the "Type" of the move multiplies its Base Power based on the Type of the attacking Pokemon (the "STAB" effect) or the defending Pokemon (in/super-effectiveness). Sometimes a move may even deal no damage due to these modifiers! For example, a Ground-type Pokemon is immune to Electric-type moves and a Ground-type move deals no damage to a Pokemon with an active Air Balloon item. Further, since many of these multipliers can apply to the same move and all operations/values are stored and computed as integers, the order of operations matters, as we illustrate in Section 3.4.

In terms of environmental mechanics, we implement weather and traps/hazards (introduced in Gen 2), environment (from Gen 3), and terrain (from Gen 7).

3.3 Data

The most time-consuming challenge in the project is to correctly implement the logic of dozens of idiosyncratic mechanics while maintaining correct interactions between every pair of mechanics, but perhaps the more interesting challenge is to structure the system to handle similar mechanics in as large and as many groupings as possible. For the latter, we need data on the mechanics, many of which comes from different sources and all of which needs to be formatted for both our understanding and the engine's processing.

3.4 Useful examples

(mostly illustrating importance of order of operations)

3.5 Miscellaneous mechanics

(Truncation, PRNG, sorting)

3.6 Monte Carlo Tree Search agent

(opponent(s) design, timing methodology)

4 Results

(Comparison of run-times for toy examples and MCTS agent)

5 Conclusion

(TBD)

References

[HL19] Dan Huang and Scott Lee. A self-play policy optimization approach to battling pokémon. In *2019 IEEE Conference on Games (CoG)*, pages 1–4, 2019.

[Lin19] Norstrom Linus. Comparison of artificial intelligence algorithms for pokémon battles. Master's thesis, Chalmers University of Technology, 2019.

[LT17] Scott Lee and Julian Togelius. Showdown ai competition. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 191–198, 2017.

[pkm] Pokemon. `https://www.pokemon.com/us`. Accessed: 2025-01-28.

[RP20] Sebastian Risi and Mike Preuss. From chess and atari to starcraft and beyond: How game ai is driving the world of ai. *KI - Künstliche Intelligenz*, 34, 02 2020.

[sho] Pokemon showdown. `https://pokemonshowdown.com/`. Accessed: 2025-01-28.

[SRLR20] David Simões, Simão Reis, Nuno Lau, and Luís Paulo Reis. Competitive deep reinforcement learning over a pokémon battling simulator. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 40–45, 2020.

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