CPSC 335 Algorithm Engineering

(Spring 2023)

Project 2

Group 8 from Project 1

Team Members:
Abdulrahman Al-Yazidi
Brandon Nguyen
Jeremy Braneke-Hite
Josh Tamara
Tung Nguyen

Project Repository: https://github.com/Jtamara/CS335-group-8-/

Abstract

As part of this project, we are required to implement a command line game. We chose to create a simple Pokémon simulator that emulates Pokémon battles from the first generation of Pokémon games.

Pokémon is a Japanese media franchise (The Editors of Encyclopaedia Britannica, 2023) that began with a pair of games called Pocket Monsters Red and Pocket Monsters Green (The Editors of Encyclopaedia Britannica, 2023). These two Game Boy (The Editors of Encyclopaedia Britannica, 2023) games are often referred to by fans as the "Generation I" (Generation, 2023).

The formulas (and more generally, all facets of game mechanics) used in Pokémon games have been reverse engineered by the fan community and can be found across fan sites like Bulbapedia, which is where we will be referencing from.

The reason we chose to create a Pokémon simulator is not just because of its sheer popularity as the highest grossing media franchise in the world (Buchholz, 2021), but because games from the core series are all zero-sum and turn-based (Mills, 2018). This allows us to imagine (since the use of a tuple to simulate movement through a tree was sufficient given the project scope) the game states of Pokémon battles as game trees, and apply a tree search to find optimal moves for the machine to play (Mills, 2018).

Scope

Given the time constraints, we needed to limit the scope of the project to keep it reasonable, including eliminating all game mechanics not tied to the Pokémon battle system.

Even for game mechanics that tie directly into battles, we have had to limit the scope due to how incredibly complex the system is. For one, there are in-game items that can be used and/or held by Pokémon (Item, 2023); players may also switch between different Pokémon in your team in the midst of battle (Pokémon battle, 2023). Pokémon moves may also induce one or more of over 50 status conditions (the probability of which is dynamically determined, and depend on a plethora of factors) which fall into three categories: non-volatile, volatile, and volatile battle, each with their own set of rules and characteristics (Status condition, 2023). Even for moves that only deal damage, there are mechanics like speed (Stat, 2023) and priority (Priority, 2023) which affect turn order; some of them also deal damage over multiple turns (Petal Dance (move), 2023) or force the user to skip turns (Solar Beam (move), 2023).

As such, all the aforementioned mechanics have been determined to be beyond the scope of this project, since these introduce a huge amount of complexity, whilst not significantly affecting our approach to implementing the machine player. We will only be handling attacks that deal damage and only last one turn; there will also be no consideration for speed or priority. We will, however, be including accuracy, critical hits, effort values (EVs, which are stat modifiers), and natural variance in damage calculations.

We have also elected to choose a fairly naïve implementation for the static evaluation function in our Minimax algorithm. It looks solely at the damage dealt per turn (alternating between player and opponent for maximizing and minimizing iterations) to determine the most optimal solution, and we believe it is sufficient for our needs even if we chose to expand the project scope down the line by allowing moves with certain side effects like healing or stat modification.

General Flow of the Program

The C++17 source code is provided alongside CMake files to allow building from source. We have also provided a compiled binary named `pokemon_simulator.exe` for testing on 64-bit Windows platform.

Upon running the program, the player will be asked to pick a Pokémon and a difficulty level. The difficulty level is a construct that we devised to showcase the capabilities of the machine player and results in different EVs spread during instantiation of Pokémon objects, such that in lower difficulty modes, the machine player has worse stats that it needs to play around. It also affects the game tree search depth, with higher difficulty modes using a deeper search. The machine player's Pokémon is selected pseudo-randomly.

Upon selection, Pokémon (and their available moves) are instantiated using a factory design pattern. Our factory classes deviate from the norm in that it produces instances of the same class instead of instances of subclasses. We chose to apply the idea behind this design pattern in order to improve maintainability. This allows extension of the program by adding new Pokémon, moves, and move pools to their respective factories without accumulating technical debt. Each Pokémon has base stats congruent to the original games; EVs are determined pseudo-randomly (within the confines of the original game mechanics) with the distribution of which being influenced by the selected difficulty level.

The player goes first, and can choose one of four moves to deal damage (in other words, reduces the health points (HP) of the opposing Pokémon) to the opponent (in this case, the machine player's Pokémon), which then brings us to the machine's turn. The machine player then uses Minimax to determine the best move to use in retaliation.

Each move has a raw power attribute, as well as accuracy (which is the likelihood that the move lands a hit), and a type. Pokémons have types as well. If a Pokémon uses a move of the same type as itself, it deals more damage. If a Pokémon uses a move that is superefficient against the opponent Pokémon's, it deals more damage. Conversely, if a Pokémon uses a move that is not effective against the opponent Pokémon's, it deals less damage. The evaluation step in our Minimax function takes into account type effectiveness, and also applies law of large numbers in the face of various sources of variance (like accuracy attributes). This produces an intelligent machine player that has a better than random chance of winning against human players, and can consistently defeat human players that have no knowledge of game mechanics.

The game continues with both the human and machine players taking turns selecting a move. When either Pokémon reaches 0 HP, the game ends and the winner is announced.

Areas for Improvement

As previously mentioned, many game mechanics have had to be left out due to time constraints. Given more time, these can be implemented for a richer gameplay experience. That being said, it would not significantly impact the approach to this problem, as we would still be using Minimax.

We also note that we are using a naïve Minimax implementation without Alpha-beta Pruning. This was a conscious design choice: given our use of depth limiting and limited project scope, we find the performance of this implementation to be sufficient and acceptable. However, if the project scope were expanded with more game mechanics, the use of Alpha-beta Pruning may become much more relevant.

In hindsight, we could also have made the machine player pick moves at random for the easiest game mode, so as to better demonstrate the capabilities of the decision-making algorithm in comparison.

References

Buchholz, K. (2021, February 24). *The Pokémon Franchise Caught 'Em All*. Retrieved from Statista: https://www.statista.com/chart/24277/media-franchises-with-most-sales/

Generation. (2023, April 7). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Generation

Item. (2023, May 3). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Item

Mills, R. A. (2018). A Deep Learning Agent for Games with Hidden Information. *Bard Digital Commons*, 18.

Petal Dance (move). (2023, February 28). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Petal_Dance_(move)

Pokémon battle. (2023, January 13). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9mon_battle

Priority. (2023, February 16). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Priority

Solar Beam (move). (2023, April 9). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Solar_Beam_(move)

Stat. (2023, March 22). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Stat#Speed

Status condition. (2023, May 5). Retrieved from Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Status_condition

The Editors of Encyclopaedia Britannica. (2023, May 9). *Pokemon*. Retrieved from Encyclopædia Britannica: https://www.britannica.com/topic/Pokemon-electronic-game