Near Peer Combat Simulator
Hayden Burger, Corinne Desroches
David Lee, John Tyler

## Executive Summary

**Problem Statement:** Which Pokémon is the best, and can we determine it?
Ultimately, the goal of this project was to take a scientific approach to a fun topic and see if we could derive some insights. Pokémon, as a franchise, is a combat adventure simulator. It combines battle tactics with a dose of randomness and strategy to create a fun strategy-dense game with an emphasis on fun and discovery. Unfortunately, the full scope of the franchise leads to a high complexity that is just too big to solve within the time allotted for this project. A full battle involves teams of 6 Pokémon chosen from 1,025 available, and then adds layers of turn order, game items, leveling choices, move choices, etc. To answer our question in time, we had to make a few assumptions.

### Simplifications

Naturally, one of our first steps was to create simplifications that did not take too much away from the overall goal. First, we chose Generation 1 Pokémon, the first set of Pokémon released, as these are the most recognizable and most repeated Pokémon in the games.  Additionally, it would be the ones that are most familiar to an average audience. Second, we fixed all Pokémon to be level 1. Level 1 Pokémon have the fewest moves available, which condensed the scope of possible move sets per Pokémon. Third, we ignored the idea of PP, which limits the number of times a Pokémon can use a particular move. This could be added in future iterations of this work. Lastly, we assumed our simulation running through 1000 iterations would be sufficient to see who would rise to the top. We chose not to go over 1000 iterations as running this took 2.5 hours on its own. Furthermore, it needed to be redone when a bug was found. Python also has a quirk that limits its computation to one core at a time without installing certain packages, so limiting the iterations to 1000 was essential to developing results that could be rapidly checked and reran if necessary. For accuracy to the games, we did elect to use the full generation 1 damage equation for battles, accounting for every variable that could affect the damage of a move.

### Data Collection

Data collection was the first order of business, and included frequent searches of Kaggle.com, serebii.net, Pokemondb.net and bulbapedia.bulbagarden.net. We collected four data setss:

1. Pokemon.csv
   - Kaggle.com dataset scraped by user 'ROUNAK BANIK' from http://serebii.net/
2. MovesetGen1.xlsx
   - Pokemondb.net move set cut and paste into
     Excel from https://pokemondb.net/move/generation/1
3. Moveset.csv
   - Kaggle.com dataset scraped by user 'KEITH SANDERS' from https://pokemondb.net
4. Moveset_per_pokemon.csv
   - We cut and pasted from https://bulbapedia.bulbagarden.net/wiki/Bulbasaur_(Pok%C3%A9mon)/Generation_I_learnset#By_leveling_up and combined by hand into a .csv file for further processing.

**Data Processing**
Much of this was done using the Pandas Python library, and consisted of formatting string data by removing non-typable characters, fixing spelling differences, normalizing different data, confirming veracity, replacing infinity characters, replacing null values, etc. We had a lot of errors between the datasets that had to be manually corrected such as the misspellings. After cleaning the data, we ensured that all information reflected generation 1 Pokémon and their moves. One issue we encountered was that the types assigned to Pokémon from later generations such as 'fairy' were left in and not fixed. We realized this late into our work and thought this would not skew our results by a large factor. Then the multiple sources of data were brought together into a singular Pandas DataFrame.

**Class Creation**
The class contains easily referenceable ways to call every major stat of each Pokémon (speed, attack, defense, special attack, special defense, hit point, type advantages, etc). It provides us with a function for each Pokémon to take normal and confusion damage in battle, choose moves, use moves, take status effects, and reset everything back to the original values at the end of a battle. It was a massive lift and created the modularity needed to tackle the project in a reasonable timeline.

**Running a Battle, and Monte Carlo Simulation**
This set of code allowed an actual battle to happen and references the class constantly for the mechanical happenings. Multiple battles are then combined into a single 151*151 battle run, and repeated in a Monte Carlo simulation to form a matrix that could be used for data analysis. The order of an individual battle can be seen in the flow charts below.

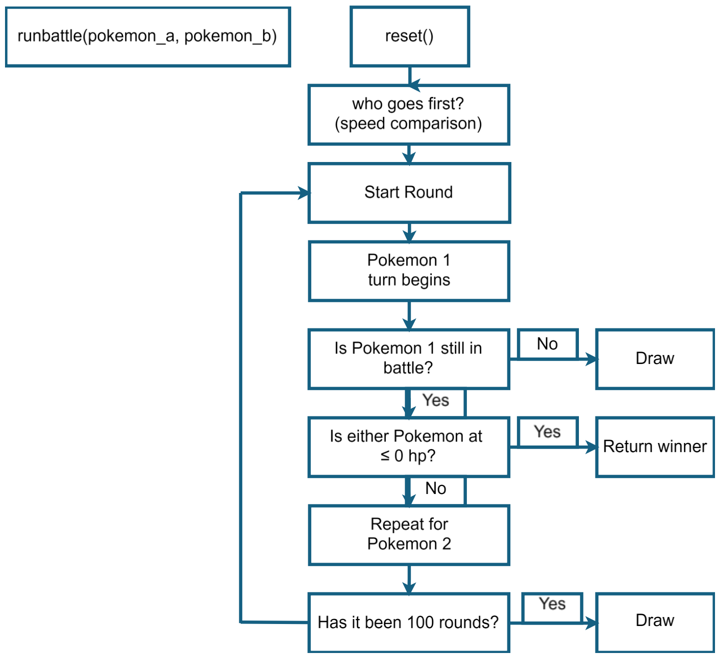

runbattle(pokemon_a, pokemon_b)
reset()
who goes first? (speed comparison)
Start Round
Pokemon 1 turn begins
Is Pokemon 1 still in battle?
No
Draw
Yes
Is either Pokemon at ≤ 0 hp?
Yes
Return winner
No
Repeat for Pokemon 2
Has it been 100 rounds?
Yes
Draw

Figure 1: Process of a typical battle

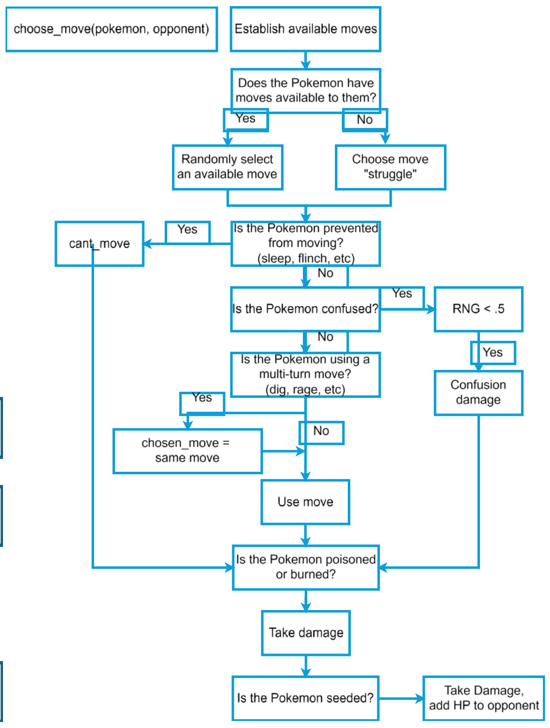

choose_move(pokemon, opponent)
Establish available moves
Does the Pokemon have moves available to them?
Yes
No
Randomly select an available move
Choose move "struggle"
cant_move
Yes
Is the Pokemon prevented from moving? (sleep, flinch, etc)
No
Is the Pokemon confused?
Yes
RNG < .5
Yes
Confusion damage
No
Is the Pokemon using a multi-turn move? (dig, rage, etc)
Yes
chosen_move = same move
No
Use move
Is the Pokemon poisoned or burned?
Take damage
Is the Pokemon seeded?
Take Damage, add HP to opponent

Figure 2: Illustration of the choose_move function

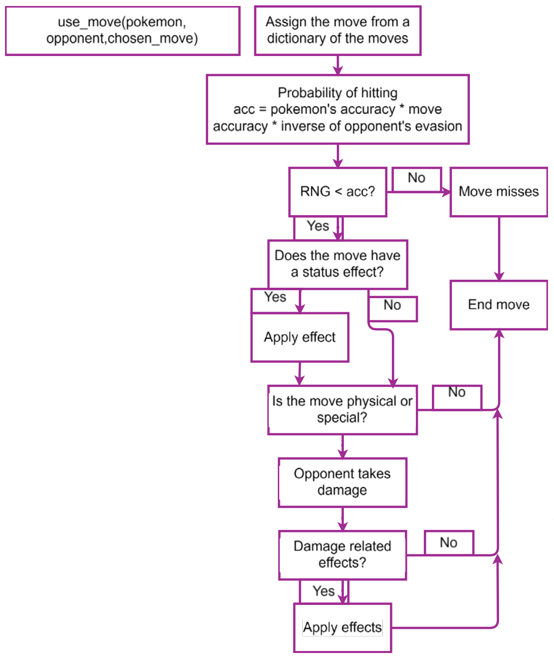
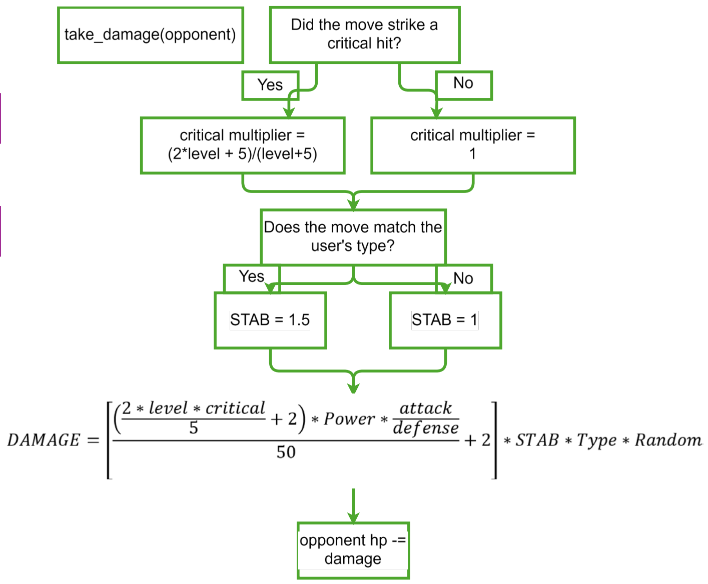

use_move(pokemon, opponent,chosen_move)
Assign the move from a dictionary of the moves
Probability of hitting acc = pokemon's accuracy * move accuracy * inverse of opponent's evasion
RNG < acc?
No
Move misses
Yes
Does the move have a status effect?
End move
Yes
No
Apply effect
Is the move physical or special?
No
Opponent takes damage
Damage related effects?
No
Yes
Apply effects

Figure 3: Illustration of the use_move function

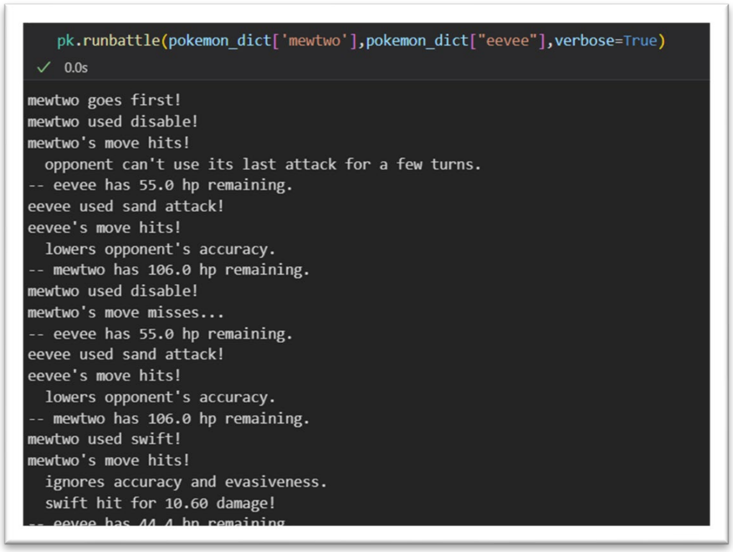

take_damage(opponent)
Did the move strike a critical hit?
Yes
No
critical multiplier = (2*level + 5)/(level+5)
critical multiplier = 1
Does the move match the user's type?
Yes
No
STAB = 1.5
STAB = 1
opponent hp -= damage

$$DAMAGE = \left[ \frac{\left(\frac{2 * level * critical}{5} + 2\right) * Power * \frac{attack}{defense}}{50} + 2 \right] * STAB * Type * Random$$

Figure 4: Flow of the take_damage function

**Battle Verification and Debugging**

To witness the results of the battle and verify all conditions and moves were behaving appropriately, we incorporated verbose statements into the class and battle simulation functions. This allowed us to catch many bugs before the final project was sent through 1000 runs of the Montecarlo simulation. As seen in the image below, we can see how each battle played out move by move.



```
pk.runbattle(pokemon_dict['mewtwo'],pokemon_dict["eevee"],verbose=True)
✓ 0.0s
mewtwo goes first!
mewtwo used disable!
mewtwo's move hits!
  opponent can't use its last attack for a few turns.
-- eevee has 55.0 hp remaining.
eevee used sand attack!
eevee's move hits!
  lowers opponent's accuracy.
-- mewtwo has 106.0 hp remaining.
mewtwo used disable!
mewtwo's move misses...
-- eevee has 55.0 hp remaining.
eevee used sand attack!
eevee's move hits!
  lowers opponent's accuracy.
-- mewtwo has 106.0 hp remaining.
mewtwo used swift!
mewtwo's move hits!
  ignores accuracy and evasiveness.
  swift hit for 10.60 damage!
-- eevee has 44.4 hp remaining
```

Figure 5: Verbose statements from a battle between Mewtwo and Eevee. Here you can see each move selected, if it hits or not, the effect or damage caused, and the remaining hp of the two Pokémon.

**Data Analysis**
All this work needed a visual representation to make it palatable, and that came in the form of bar charts, sorted lists, and comparison charts that collected Pokémon by type.

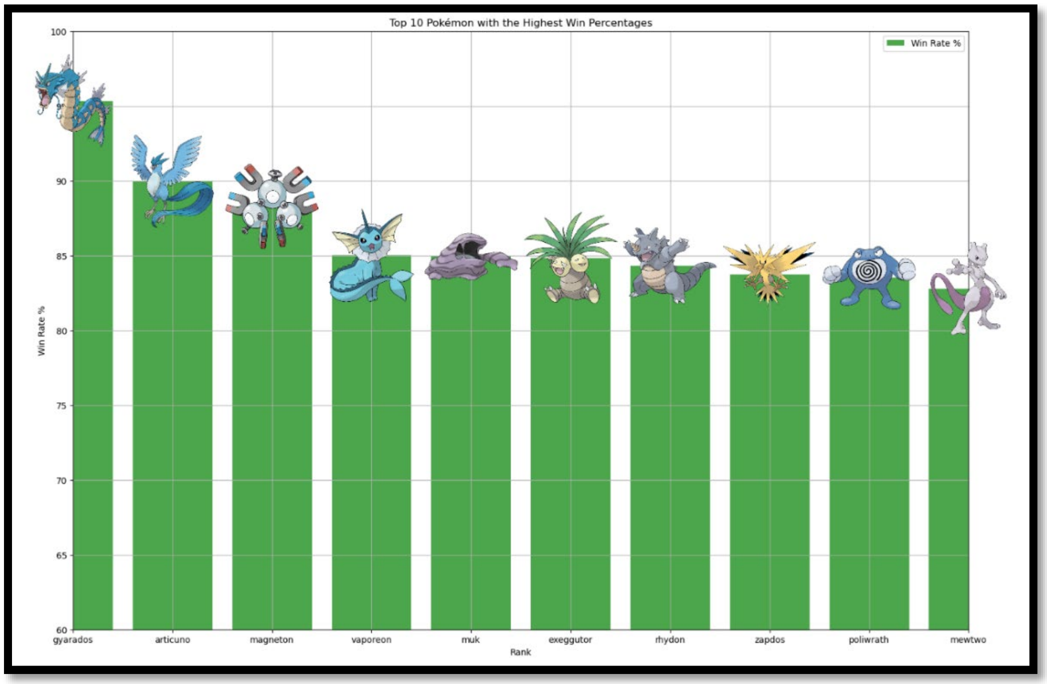
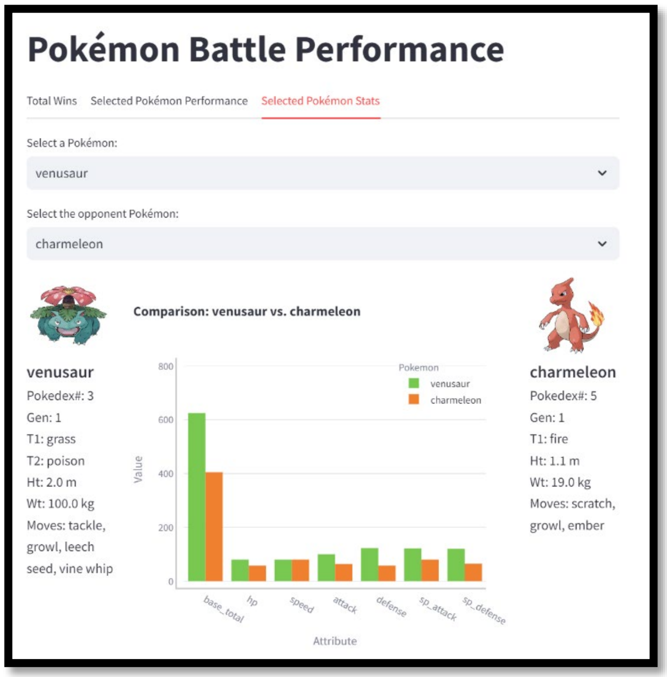


Chart 1: The top 10 Pokémon with largest win rates.

**Wide Accessibility**
Further expansion of the project was accomplished by creating a Streamlit app, and the associated Github platform necessary to create it. This app allows the user to see and explore the results of our work. Here we allow the user to compare the win rate against any specific Pokémon attribute (speed, hp, defense, etc.). We also enable the user to compare two Pokémon side by side. This provides the user a way to view their stats, their win rate in our simulation, and we allow them to run a battle in real time and see the print out from our verbose statements.



Select a Pokémon:
venusaur

Select the opponent Pokémon:
charmeleon

venusaur
Pokedex#: 3
Gen: 1
T1: grass
T2: poison
Ht: 2.0 m
Wt: 100.0 kg
Moves: tackle, growl, leech seed, vine whip

charmeleon
Pokedex#: 5
Gen: 1
T1: fire
Ht: 1.1 m
Wt: 19.0 kg
Moves: scratch, growl, ember

Figure 6: Snapshot of the Streamlit app.