**Description**

This program an interactive Pokémon battle simulator. You can pick the 6 Pokémon that you want for your team, as well as their moves and EV distribution, and then play against the computer in a battle. All Pokémon through Generation 4 will be included, as well as moves through all of the generations.

**Competitive Analysis**

There are two similar projects to compare against:

1. **Pokemon Simulator in Python** by christophercao1 (<https://github.com/christophercao1/Pokemon-Simulator-in-Python>): This project is a single screen split into two sides that allows a single user to battle two pokémon against each other. The moves are preset for each pokémon and you cannot enter other viable moves that the pokémon can learn. It is limited to only first generation pokémon. There is a box in the middle that tells you what move each pokémon used and how much damage they dealt. There is no actual “battlefield” graphic.
2. **Pokemon Version 112** (<https://youtu.be/dL8FJTb4UsA>): This simulator has a fairly expansive teambuilder that allows you to form multiple battle teams and add pokémon to each team as well as their moves. There is music for the actual battle, still sprites, and instead of playing both pokémon yourself, you can play against the computer.

This *Pokémon Battle Simulator* will improve upon the first two in that there are better graphics/design, with animated sprites and a select screen that allows for the user to click separate boxes to contain each pokémon instead of just one select screen for a team, as the second project features. This project will also have different levels of the computer to play against instead of just either playing against yourself manually, or playing against one level of the computer. There will also be a wider variety of pokémon offered for play in general.

**Structural Plan**

object Pokemon

* imports Move object (below) so pokémon can use the moves in the attack method
* Attributes: name, type, level, EV list, moves list, base stats
* Methods: attack (useMove?), reduce/increaseStats by x stages (for in-game move effects), is[StatusCondition], isProtect (protected?), + typical get attribute methods

object Move

* Attributes: name, type, category (physical, special, status), power, accuracy, pp, priority, added effects (e.g. status conditions, whether it increases/reduces a Pokemon’s stats, etc.)
* Methods: typical get attribute methods

object Player

* Attributes: team (6 pkmn)
* Methods: switch (switch out) a pkmn

object EasyOpponent

* inherits from Player
* in-battle algorithm stuff probably

object HP bar

* Attributes: level, hp, cx, cy (center x, center y for blitting)
* Methods: draw

object Battle HP bar

* inherits from HP bar; adds attributes pkmn and Max HP + overrides draw function to acct for what color bar should be when hp drops below some threshold

Damage Calculator.py

* some damage calculator thing for the computer on hard difficulty; takes in two pokémon and some moves as inputs

Status Calculator.py

* same as damage calc, but for status

Pokemon csv file

* csv file so program can unpack it; contains all pokémon information for object Pokemon attributes

Move csv file

* csv file so program can unpack it; contains all move information for object Move attributes

In addition, there will be a folder for images, a folder for all the classes to go into (plus spreadsheets, imgs needed there), and the init/run function will go in the top directory. Also, perhaps a folder for fonts.

**Algorithmic Plan**

Since this program involves the user playing against the computer, a plan for the algorithm for the computer’s strategy against the player will be roughly as follows:

Easy difficulty (by TP2):

* Pokémon in the team are entirely randomly selected, and will be at least but not above the levels of the pokémon in the player’s team.
* No switching out. Send out pokémon as they come.
* If there’s a super effective move the pokémon has against the player’s pokémon, use it. Otherwise, randomly select a move from the pokémon’s movepool to use.
* When a pokémon faints, randomly select an unfainted pokémon from the team to send out.
* Rinse and repeat.

Medium difficulty:

* Everything in the easy difficulty, except remove any restrictions.
* Pokémon in the team will still be randomly selected, but also take into consideration the player pokémons’ levels (e.g. if the player has an all-100 level team, the computer will have a team ranging anywhere from lv. 95-100).
* Whenever the opponent (the player) sends out a pokémon, save that pokémon to a dictionary with items comprising of an empty list. Likewise, whenever the player’s pokémon uses a move, save that move to the index that pokémon occupies in the dictionary. This way, the computer can use this dictionary of the player’s team for future reference.
* If the type of the player’s pokémon is strong against the computer’s pokémon, the computer will look through its own team to see if there is a pokémon that is strong against the player’s pokémon. Switch out if the current pokémon the computer has out is over 30% in HP; otherwise there’s not really a point.
* Once a pokémon faints, the computer will search through its team for a type that is super-effective against the current pokémon on the player’s side of the field. If both pokémon have fainted that turn, just send out a random pokémon.
* Occasionally use Protect to scope out what the player will do, and to fill up the dictionary.

Hard difficulty (lower priority? will do after finish uploading graphics):

* Everything in the medium difficulty, except remove any restrictions. (Also improve on any shortcomings the medium algorithm might have, for instance, revise the strategy in the fourth bullet point.)
* Pokémon in the team will be randomly selected, but the team will be as diverse as possible (e.g. no monotype or mostly-monotype Rock/Water/Fire/etc. team). The team’s levels will also always be higher than player’s pokémon team’s levels, if possible (e.g. if the player has an all-100 level team, the computer will have an all-100 level team; if the player has a ~90 level team, the computer will have a 92+ level team or something).
* The computer has knowledge of the base stats of the player’s pokémon (applying only to pokémon in the computer’s dictionary of the player’s team). So, for instance, if a certain pokémon has a higher Sp. Atk stat than an Atk Stat, the computer will take that into consideration and see if any of its own pokémon with high Sp. Def stats will deal alright with the opposing pokémon. (Priority goes to whether or not that pokémon’s typing is a bad matchup, though.)
* If the player’s pokémon is a good type matchup to the computer’s pokémon, the computer’s pokémon will use Protect, if possible, to scope out the player’s pokémon’s move and to fill up its dictionary. Proceed from there.
* If the computer’s in-play pokémon has a super-effective move against the player’s in-play pokémon, it will calculate how much damage it does to the player’s in-play pokémon using that move, as well as how much damage a super-effective known move from the player’s pokémon will do to its own pokémon. (If no known moves are super-effective or there are just no known moves in general, pick a generic common high power, say ~90, and calculate the damage from that.)
  + If the computer’s pokémon will faint and the computer’s pokémon is slower, then the computer will look through its own team for a switch.
  + If the computer’s pokémon will faint and the computer’s pokémon is faster, the computer checks if the damage output from its own pokémon brings the opposing pokémon below 20%. If so, complete the move. Else, consider a switch out.
  + If the computer’s pokémon will not faint but will fall below 20% and the computer’s pokémon is slower, consult the dictionary to see if other pokémon in the player’s party are weak to the current pokémon. If so, consider switching out. If not, complete the move; then repeat this section’s algorithm thing.
  + If the computer’s pokémon will not faint but will fall below 20% and the computer’s pokémon is faster, complete the move.
  + etc. etc. for all cases
* If, during a turn, both the player and the computer’s pokémon have fainted, the computer will check the dictionary of the player’s pokémon. Take into account stats, typings, etc. and make as much of an “informed” decision as possible.
* etc etc

Probably will add more as I go.

**Timeline**

11/20/18 (TP1): By this point, finish final selection screen graphic fiddly things.

11/25/18: By this point, have a working “full game”: battle mechanics work; easy difficulty algorithm finished, so that MVP is achieved (? unless MVP requires medium difficulty? Otherwise, finish MVP project by 11/27/18).

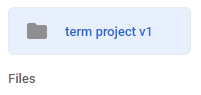
11/26/18: By this point, implement medium difficulty algorithm? (Dunno how long it will take to build off easy algorithm.)

11/28/18 (TP2): Try to have finished medium difficulty by this point.

The Rest Of The Time Until TP3: Finish uploading graphics & information for pokémon through Generation 4. (493.) Then do hard difficulty algorithm.

**Version Control**

I will be using Google Drive (and potentially GitHub, once I figure out how to use it) for version control, and upload a version about every 3-4 days or so starting from 11/19/18.



The image seen above is the project uploaded to Google Drive in advance of submitting TP1.

Have also created a GitHub since then for version control.

**Module List**

* Pygame

**TP 3 Update**

* Will not be including all pokémon through Generation 4; all of Generation 1 is implemented, however.
* different CPU levels
* added in bars in the battle screen to indicate how many battle-able pokémon the player and cpu have left, so the player has an idea of where in the battle they are at
* added message text to the screen (e.g. “[Pokemon] used [Move]!” or “[Pokemon] is paralyzed and can’t move.”)
* implemented suggested EV spread for player pokémon once you’ve inputted some moves