Python Pokemon Battle Simulator

Report for Final Project

EE 3326 Numerical and Scientific Data Analysis Using Python

Submitted by

Hunter Chopskie

Ingram School of Engineering

Texas State University

2 December 2022

ABSTRACT

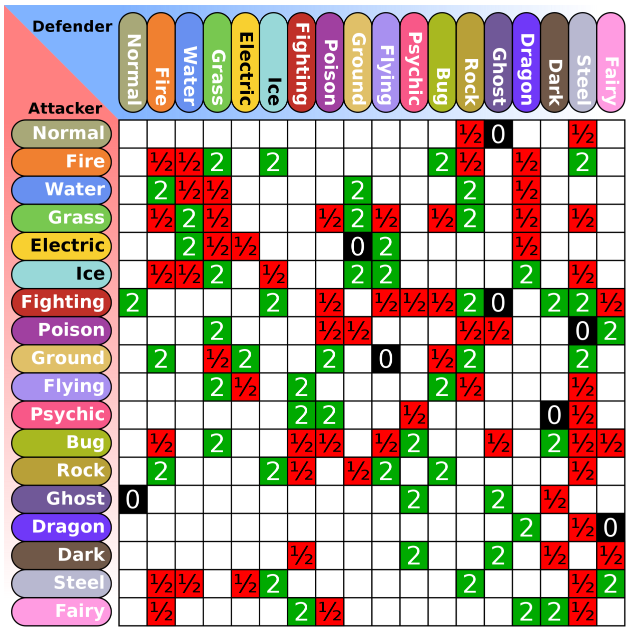
The purpose of this project was to construct a Python program that simulates a Pokemon battle between two players. Pokemon is a media franchise created by Nintendo and Game Freak in 1996. This project focused on simulating the main component of their video game series, a battle that takes place between two opposing players. In order to accurately simulate the game, several features and components had to be well researched, understood, and implemented. This includes acquiring game data from existing sources to implement it into the program, learning new features in Python to modify the program, and having a strong understanding of the game's mechanics. This project utilizes Python's data structures and several data analysis tools to best accomplish the goal of accurately simulating the game.

BACKGROUND

In order to fully understand the program's processes and objectives, one must understand the core mechanics of the battle between two players. Battles are a turn-based game, where each turn of the game, both players choose an action. In a full battle, both players are given six pokemon on their team, and battle using one pokemon at a time. Both players choose their own actions each turn, choosing to either switch their pokemon to a different one on their team, or choose a move to inflict damage on the opposing player's pokemon's health, also known as hit points, or HP for short. When a pokemon is depleted of its HP, it is knocked out, rendering them unusable for the rest of the battle. A battle ends when one player has their whole team of pokemon knocked out, with that player losing, and the other player being declared the winner. There are over 900 pokemon in the franchise, each with their own unique combination of name, type, stat distribution, and moves. These properties are explained below:

Names: Every pokemon has its unique name, that lets players easily distinguish between them.

Types: Types are properties that are applied to every pokemon, as well as their moves. They are the most important part of the battle, as they are what influence player decision and strategy. All pokemon have either one or two types applied to them, and all moves have their own singular type. Every type has its own set of strengths and weaknesses over other ones, which can be best equated as a higher-scale version of a simple rock-paper-scissors game. In the game, there are 18 types, each with their own effectiveness against all other types. A chart depicting type effectiveness is provided below:



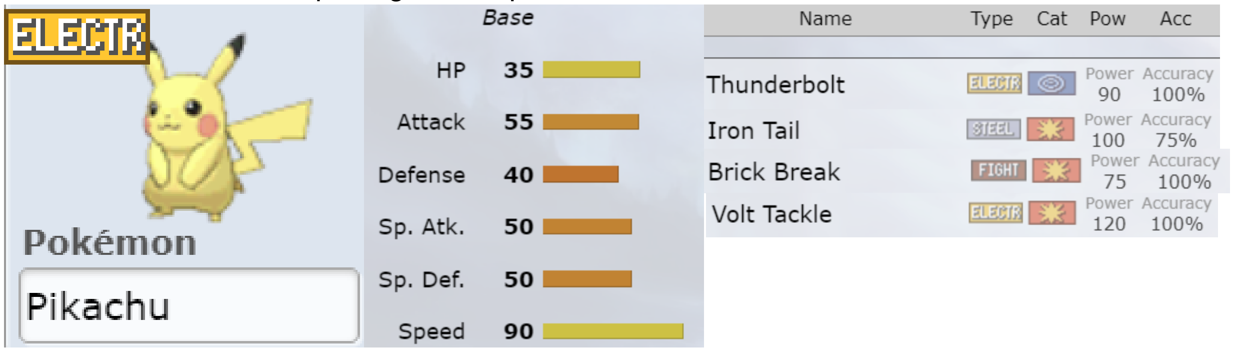
Defender Attacker	Normal	Fire	Water	Grass	Electric	Ice	Fighting	Poison	Ground	Flying	Psychic	Bug	Rock	Ghost	Dragon	Dark	Steel	Fairy
Normal													½	0			½	
Fire		½	½	2		2						2	½		½		2	
Water		2	½	½					2				2		½			
Grass		½	2	½				½	2	½		½	2		½		½	
Electric			2	½	½				0	2					½			
Ice		½	½	2		½			2	2					2		½	
Fighting	2					2		½		½	½	½	2	0		2	2	½
Poison				2				½	½				½	½			0	2
Ground		2		½	2			2		0		½	2				2	
Flying				2	½		2					2	½				½	
Psychic							2	2			½					0	½	
Bug		½		2			½	½		½	2			½		2	½	½
Rock		2				2	½		½	2		2					½	
Ghost	0										2			2		½		
Dragon															2		½	0
Dark							½				2			2		½		½
Steel		½	½		½	2							2				½	2
Fairy		½					2	½							2	2	½	

In a battle, a move of a single type is used against a pokemon of one or two types. The move can be thought of as the "attacker" label in the chart, while the pokemon being on the receiving end of the move can be thought of as the "defender". When determining type effectiveness, The move's type is taken into consideration against the defending pokemon's type. The values in each combination square tells the game how to implement a multiplier into the main damage calculator, where blank squares are depicted as "1", and having no strong or weak multiplier associated with it. For example, a water type move used against a fire type pokemon will be super effective, therefore it will multiply the damage dealt by two. A fire move used against a water type pokemon will not be very effective, therefore it will multiply the damage dealt by 0.5. When a move is used against a pokemon with multiple types, the move's effectiveness against the pokemon's first type and its second type are multiplied together. For example, if a fire type move is used against a pokemon with a type combination of water and grass, it is considered "neutral", because the strong effectiveness against grass and the weak effectiveness against water cancel out. Some types have immunities, which makes any pokemon containing that type immune to certain types of moves. For example, any pokemon with the flying type is immune to all ground moves, regardless of the pokemon's secondary type. In damage calculation for immunities, the damage is multiplied by 0, therefore no damage is done. Types are the basis that the strategy and player decision of pokemon battles stems from, therefore it must be understood on how they are implemented into the program. Further detail on how this is translated into the Python program is in the “Battle Procedure” section.

Stat Distribution: Every pokemon has their own set of base statistics defined by the game that tells their strengths and weaknesses, and overall how powerful they are. The greater the total of base stats, the stronger a pokemon generally is. Base stats are divided into six categories: HP, Attack, Defense, Special Attack, Special Defense, and Speed. Every pokemon has an integer value associated with each base stat, ranging from 5 to 255. HP is the pokemon’s health, and tells how much damage it can take. Attack tells how much damage a pokemon does with physical moves, and defense tells how much damage a pokemon receives from physical moves. Special attack and special defense are the exact same as attack and defense, but only from special moves. Distinction between physical and special moves is explained in the “Moves” section. Lastly, speed tells how fast a pokemon is. In every turn of a battle, the pokemon with the higher base speed moves first.

Moves: Moves are the methods from which damage is dealt. All pokemon contain four distinct moves, each with their own unique combination of name, type, accuracy, base power, and category. A move’s type tells how effective it will be against the pokemon on the receiving end of the move, its accuracy tells how accurate it is at hitting the opposing pokemon, and its base power tells how strong it is in the damage calculator. Lastly, the moves category tells if it will be using the pokemon’s physical attack or special attack. A move can be thought of as physical or special depending on the description of the move, and how it is depicted. For the purpose of this assignment, further elaboration is unnecessary. Instead, all that should be known is that every move in which damage is dealt is either physical, which uses the user’s physical attack stat, or special, which uses the user’s special attack stat. Lastly, there are moves whose category is “status”. These moves have a very high variety, as they have a multitude of effects, such as raising the users’ stats, lowering the opponent’s stats, and much more. For the purpose of this project, status moves do occasionally appear in movesets, but they are not implemented to do anything, as implementing them all would extend the process of developing the program greatly.

Summary: Putting all of these traits together allows us to explain the pokemon on each player's team, giving us the basis on how to implement it in the program. An example of a pokemon with these all these features put together is provided below:



ELECTR

Pokémon

Pikachu

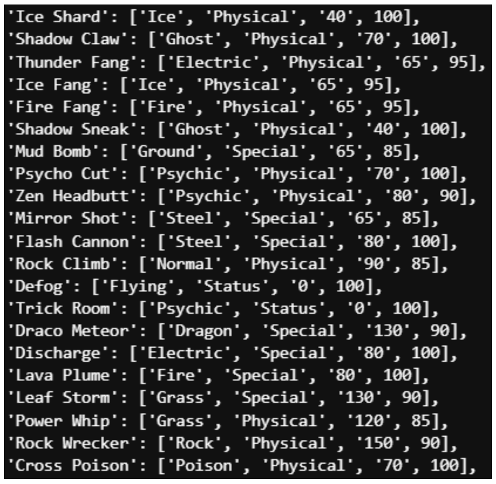
	Base
HP	35
Attack	55
Defense	40
Sp. Atk.	50
Sp. Def.	50
Speed	90

Name	Type	Cat	Pow	Acc
Thunderbolt	ELECTR	◎	Power 90	Accuracy 100%
Iron Tail	STEEL	✸	Power 100	Accuracy 75%
Brick Break	FIGHT	✸	Power 75	Accuracy 100%
Volt Tackle	ELECTR	✸	Power 120	Accuracy 100%

In this example, the pokemon "Pikachu" has a single type of electric, with base stats HP: 35, Attack: 55, DefenseL 40, Special attack: 50, Special Defense: 50, and Speed: 90. The first move it has is "Thunderbolt", an electric type attack of category: special, with 90 base power, and 100% accuracy. Next, it has "Iron Tail", a steel type attack of category: physical, with 100 base power, but only 75% accuracy. Third, it has "Brick Break", a fighting type attack of category: physical, with 75 base power, and 100% accuracy. Lastly, it has "Volt Tackle", an electric type attack of category: physical, with 120 base power, and 100% accuracy. Now that we have our background, we can discuss how the game mechanics are implemented in Python.

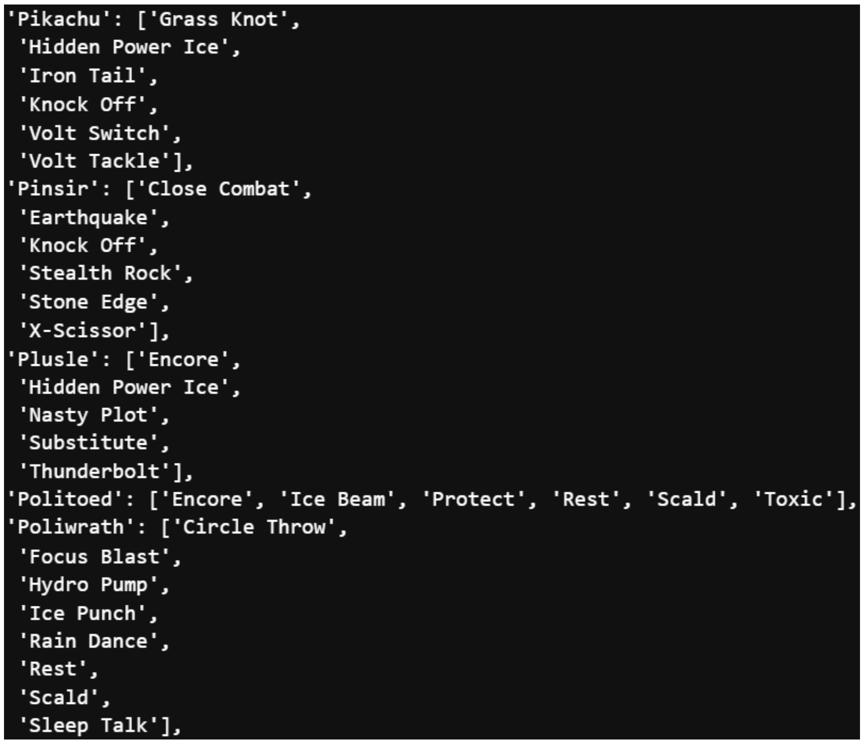
DATA INITIALIZATION

The first step in creating a program that allows us to play the game is to obtain all the data about every pokemon, and all the move data. For the purpose of this project, all of this data will be organized into lists and dictionaries. There are resources containing spreadsheets of all this data, which I obtained on the data resource known as Kaggle. This gave me the "updatedpokemon.csv" spreadsheet, and the "moves.csv" spreadsheet. Using the Pandas library, I extracted each column of these spreadsheets into distinctive lists. Once all these lists were created, I shorten the list containing the pool of pokemon the program picks from when generating teams. The reason for this is that many pokemon in the game go through a process known as evolution, where they change into a stronger form of themselves. For the purpose of equal battles between two players, I shortened the list of available pokemon to only contain fully evolved pokemon. This was accomplished by utilizing a column in the spreadsheet, indicating with a boolean value if the pokemon was fully evolved. Next, all the lists containing the base stats for all pokemon are modified, as the in-game damage calculator uses a modification of base stats when determining damage. This will be further explained in the section on damage calculation. After all pokemon with their stats and types were organized into lists, the moves from the “moves.csv” file were organized. This was utilized with a dictionary, where the keys are the name of the move, and the values are a list containing the move’s type, category, base power, and accuracy. A portion of this dictionary is provided below:



```
'Ice Shard': ['Ice', 'Physical', '40', 100],
'Shadow Claw': ['Ghost', 'Physical', '70', 100],
'Thunder Fang': ['Electric', 'Physical', '65', 95],
'Ice Fang': ['Ice', 'Physical', '65', 95],
'Fire Fang': ['Fire', 'Physical', '65', 95],
'Shadow Sneak': ['Ghost', 'Physical', '40', 100],
'Mud Bomb': ['Ground', 'Special', '65', 85],
'Psycho Cut': ['Psychic', 'Physical', '70', 100],
'Zen Headbutt': ['Psychic', 'Physical', '80', 90],
'Mirror Shot': ['Steel', 'Special', '65', 85],
'Flash Cannon': ['Steel', 'Special', '80', 100],
'Rock Climb': ['Normal', 'Physical', '90', 85],
'Defog': ['Flying', 'Status', '0', 100],
'Trick Room': ['Psychic', 'Status', '0', 100],
'Draco Meteor': ['Dragon', 'Special', '130', 90],
'Discharge': ['Electric', 'Special', '80', 100],
'Lava Plume': ['Fire', 'Special', '80', 100],
'Leaf Storm': ['Grass', 'Special', '130', 90],
'Power Whip': ['Grass', 'Physical', '120', 85],
'Rock Wrecker': ['Rock', 'Physical', '150', 90],
'Cross Poison': ['Poison', 'Physical', '70', 100],
```

After all the data was obtained from the spreadsheets and organized into usable data structures, the next step was to obtain movesets for each potential pokemon for a team. Because teams for both players are randomly generated by picking a sample of six pokemon from the list of potential pokemon. All pokemon in the list must have possible movesets, but instead of creating the movesets for every possible pokemon, we are able to utilize another existing resource: movepool data. In order to obtain movepool data, we must know how a resource known as Pokemon Showdown works. Pokemon Showdown is an open-source program that functions as a battle simulator for tens of thousands of players daily. Players are able to build teams and battle against each other. One feature of Pokemon Showdown is the random battle, which starts a battle between two players, giving them a random team of six pokemon, all with generated movesets. This is accomplished by creating a pool of moves for the program to choose from, typically ranging in size from four to ten. This pool of moves exists on Pokemon Showdown's main repository on GitHub. It is in the form of a nested dictionary, as it is a .json file. An entry of the dictionary is provided: "Pikachu": {"level": 90, "abilities": ["Lightning Rod", "Static"], "items": ["Light Ball"], "moves": ["Grass Knot", "Hidden Power Ice", "Iron Tail", "Knock Off", "Volt Switch", "Volt Tackle"]}. Due to the nature of the simulator I created, I did not need to consider level, abilities, or items. These would be implemented in a much higher-level battle simulator, but for the purpose of this program, they were to be disregarded. This type of entry appears for every pokemon possible, therefore I dumped it into a text file, and used the JSON library to read and parse through it. I extracted the pokemon name, and its list of possible moves to pick from, using nested dictionary comprehension. After this process, I had a new dictionary, containing every pokemon's name as the keys, and the list of their moves to pick from as the values. A portion of this dictionary is provided below:

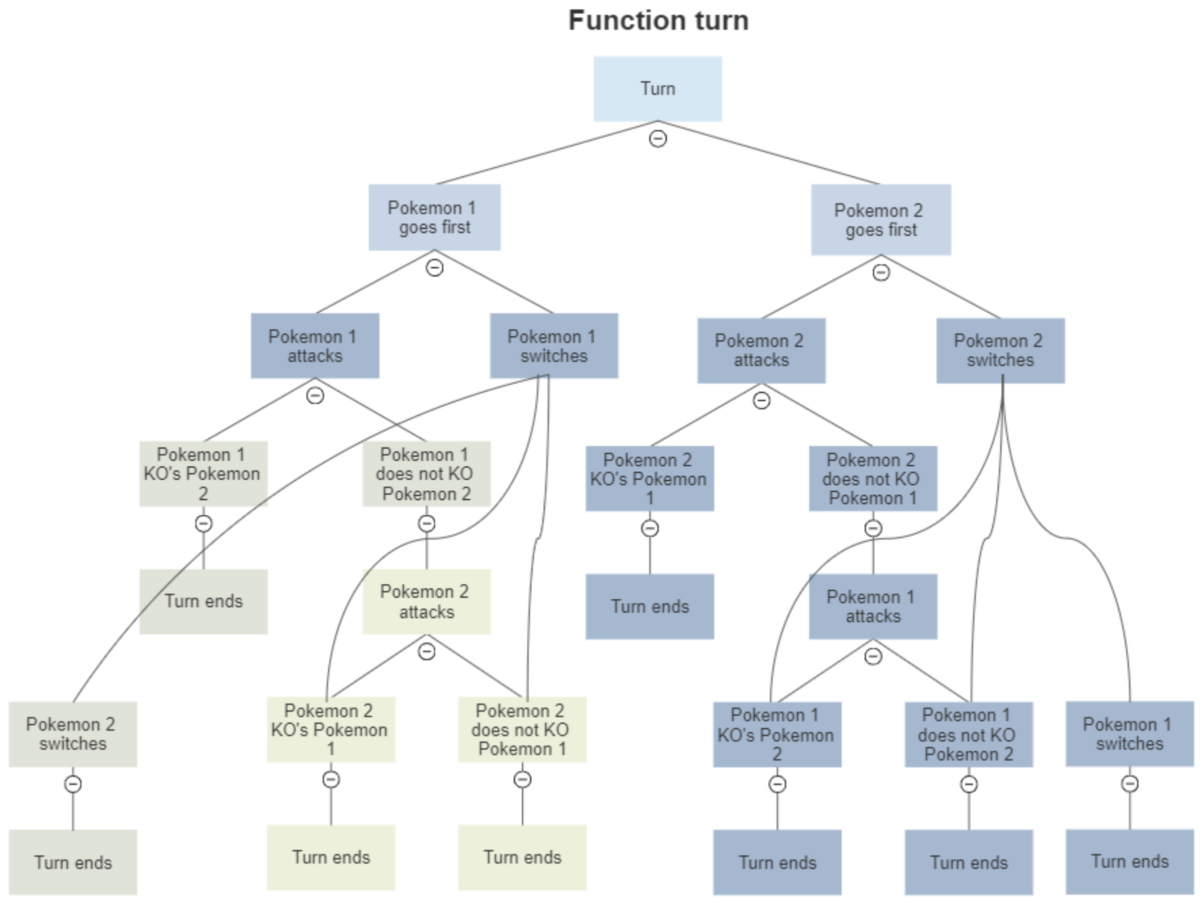


```
'Pikachu': ['Grass Knot',
 'Hidden Power Ice',
 'Iron Tail',
 'Knock Off',
 'Volt Switch',
 'Volt Tackle'],
'Pinsir': ['Close Combat',
 'Earthquake',
 'Knock Off',
 'Stealth Rock',
 'Stone Edge',
 'X-Scissor'],
'Plusle': ['Encore',
 'Hidden Power Ice',
 'Nasty Plot',
 'Substitute',
 'Thunderbolt'],
'Politoed': ['Encore', 'Ice Beam', 'Protect', 'Rest', 'Scald', 'Toxic'],
'Poliwrath': ['Circle Throw',
 'Focus Blast',
 'Hydro Pump',
 'Ice Punch',
 'Rain Dance',
 'Rest',
 'Scald',
 'Sleep Talk'],
```

BATTLE PROCEDURE

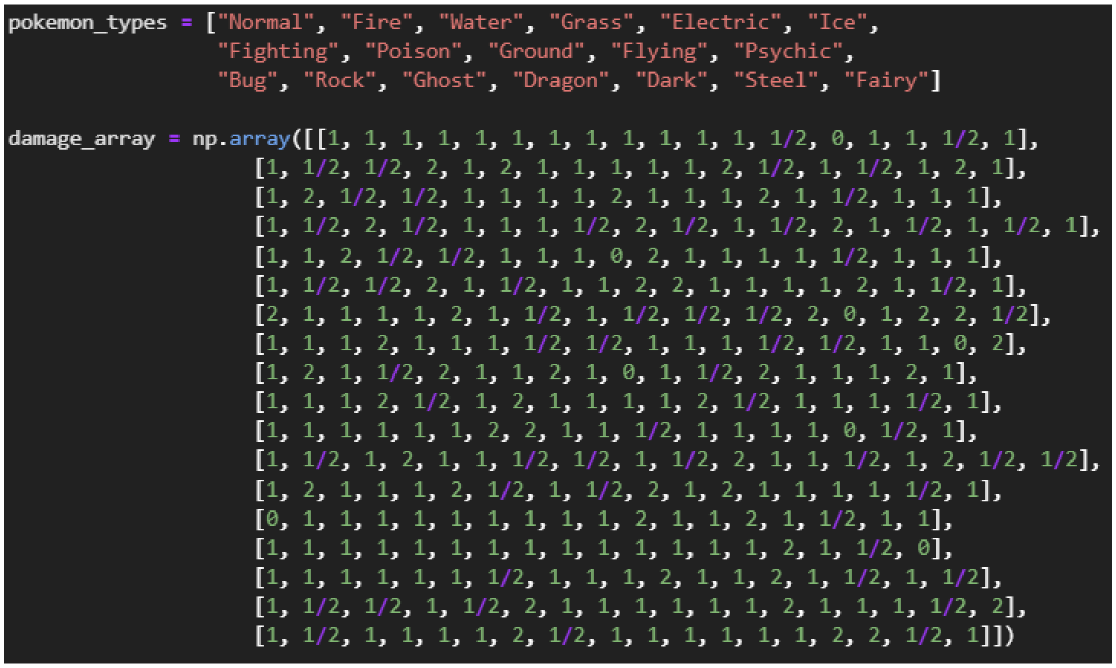
Now that all the data is obtained, analyzed, and initialized, we are able to begin the battle process. When the program starts, two teams are generated for each player, picking a random sample of six pokemon from the pool of available ones. The program organizes each player's team into dictionaries, where the pokemon names are the keys, and the values are a list containing the pokemon's raw HP that changes as damage is dealt, the pokemon's maximum HP which is used to display information to the user, and a nested list containing the four picked moves for the pokemon. The battle starts with both players sending out the first pokemon on their team. Next, the turn function, which is the main function that does most of the work, is called each turn of the battle. The remainder of the battle is done through the program's several functions. Throughout the battle, the sleep feature is used from the time library to simulate the game process and simplify the text presented to the user. Descriptions for all functions are provided below:

Function turn: The **turn** function takes both players' teams and the current pokemon as arguments, as well as the turn count. Each turn, the turn count, and both of the pokemon's moves and current HP is displayed. The program prompts the user to pick a move, using exception handling to make sure that a valid input is entered. There are several possible outcomes that can happen from each turn. First, the function calls the speed stat of both pokemon on the field to determine who goes first. If their base speed stats are the same, then it is determined by a 50/50 chance. The first player's move is used, calling the **move_used** function. This function returns a value telling if the move did damage, or did enough damage to knock out the opposing pokemon. If the opposing pokemon is knocked out, the turn ends, calling the **main_loop** function at the very end. If the first player's move does not knock out the opposing pokemon, then the slower pokemon attacks next, once again calling the **move_used** function. Regardless of if the slower pokemon does in its attack, the turn ends after it. At the end of each turn, if there is no pokemon knocked out, the dynamic HP value is reduced accordingly depending on how much damage is done. If a pokemon is knocked out during a turn, a boolean flag is raised, and its dynamic HP value in the dictionary is changed to zero. This is passed onto the **main_loop** function, which modifies the team after each turn in the event of a KO. At the very end of the function, The **main_loop** function is called at the end, allowing for the battle to continue. It should be noted that switching always takes priority and goes before any move is used. Additionally, when a player only has one pokemon remaining, they are not given the option to switch. All possible outcomes from the **turn** function are displayed in the diagram below:



<u>**Function move_used**</u>: the **move_used** function is called each time a player chooses a move for their pokemon. The arguments it takes from the **turn** function are the pokemon's name, the move's name, type, category, base power, and accuracy, the opposing pokemon's name, its dynamic HP, and its max HP. First, it displays that the pokemon uses the move chosen. Next, it

calls a function to determine if the move connected, based on the moves accuracy. If the move misses, then the function ends, returning the defending pokemon's current HP, as it was not modified. Next, it gets the index type of the move for a list of types, and the index of the defending pokemon's types. It uses these with a NumPy array containing the type effectiveness in order to get the correct damage multiplier. These actions allow us to get the vertical position on the type effectiveness chart previously shown for the move that is used, and the horizontal position for the type of the pokemon on the receiving end of the move. The array is the same as the type effectiveness chart. The array and type list are provided below:



```
pokemon_types = ["Normal", "Fire", "Water", "Grass", "Electric", "Ice",
                 "Fighting", "Poison", "Ground", "Flying", "Psychic",
                 "Bug", "Rock", "Ghost", "Dragon", "Dark", "Steel", "Fairy"]

damage_array = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1/2, 0, 1, 1, 1/2, 1],
                    [1, 1/2, 1/2, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1/2, 1, 1/2, 1, 2, 1],
                    [1, 2, 1/2, 1/2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1/2, 1, 1, 1],
                    [1, 1/2, 2, 1/2, 1, 1, 1, 1/2, 2, 1/2, 1, 1/2, 2, 1, 1/2, 1, 1/2, 1],
                    [1, 1, 2, 1/2, 1/2, 1, 1, 1, 0, 2, 1, 1, 1, 1, 1/2, 1, 1, 1],
                    [1, 1/2, 1/2, 2, 1, 1/2, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1/2, 1],
                    [2, 1, 1, 1, 1, 2, 1, 1/2, 1, 1/2, 1/2, 1/2, 2, 0, 1, 2, 2, 1/2],
                    [1, 1, 1, 2, 1, 1, 1, 1/2, 1/2, 1, 1, 1, 1/2, 1/2, 1, 1, 0, 2],
                    [1, 2, 1, 1/2, 2, 1, 1, 2, 1, 0, 1, 1/2, 2, 1, 1, 1, 2, 1],
                    [1, 1, 1, 2, 1/2, 1, 2, 1, 1, 1, 1, 2, 1/2, 1, 1, 1, 1/2, 1],
                    [1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1/2, 1, 1, 1, 1, 0, 1/2, 1],
                    [1, 1/2, 1, 2, 1, 1, 1/2, 1/2, 1, 1/2, 2, 1, 1, 1/2, 1, 2, 1/2, 1/2],
                    [1, 2, 1, 1, 1, 2, 1/2, 1, 1/2, 2, 1, 2, 1, 1, 1, 1, 1/2, 1],
                    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1/2, 1, 1],
                    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1/2, 0],
                    [1, 1, 1, 1, 1, 1, 1/2, 1, 1, 1, 2, 1, 1, 2, 1, 1/2, 1, 1/2],
                    [1, 1/2, 1/2, 1, 1/2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1/2, 2],
                    [1, 1/2, 1, 1, 1, 1, 2, 1/2, 1, 1, 1, 1, 1, 1, 2, 2, 1/2, 1]])
```

Next, the function checks to see if the pokemon receiving the move has one or two types, which affects the damage multiplier. It does this by using the .isna feature in Pandas to detect this. This is because when importing the pokemon data from the spreadsheet using Pandas, all empty cells for pokemon that did not have a second type were declared as null. After obtaining the correct damage multiplier, the function checks the category of the move. It calls the **damage_calc** function, using all necessary arguments, including the attacking pokemon's physical or special attack, its types, etc. The **damage_calc** function returns a float value, which is the damage done to the opposing pokemon. This value is truncated, as pokemon games always round down in their damage calculations. The program displays a message saying the move's effectiveness, and the percent of HP lost by the opposing pokemon. This is calculated using the pokemon's constant HP stat, which does not change. This way, the percent lost will be consistent. If the function sees that the damage done exceeds the HP that the pokemon has, the program displays that the pokemon fainted, and the function returns a unique value that the **turn** function can use to point out when a pokemon has been knocked out. Additionally, the percent lost caps out at how much percentage of HP the pokemon has. For example, if a pokemon has 9% of its HP remaining, and it is hit by a move that does 25% of its HP, it will display that it lost 9% of its HP. If the percent lost is calculated to be greater than 100%, then it caps out at 100%. If the pokemon does not faint, then the function returns the difference between the pokemon's HP and the damage done. This is used by the **turn** function to modify the pokemon's dynamic HP accordingly. Lastly, if the move category is "status" the function displays the message that status moves are not implemented, and returns the opposing pokemon's HP as no damage was dealt. This same return is done when a pokemon is immune to a move.

Function damage_calc: The **damage_calc** function is called every time a move is used, in order to determine its damage against the opposing pokemon. The arguments it takes are the move's base power, the user's attack or special attack, the opposing pokemon's defense or special defense, the move's type, the type(s) of the pokemon using it, and the damage multiplier obtained from the **move_used** function. The main formula for damage calculation in pokemon games is provided below:

$$Damage = \left(\frac{\left(\frac{2 \times Level}{5} + 2\right) \times Power \times A/D}{50} + 2 \right) \times Critical \times random \times STAB$$

In the context of this project, level is 100 for all pokemon. Power is the base power of the move, A is the attack or special attack of the pokemon using the move, and D is the defense or special defense of the pokemon receiving the move. Critical is the critical hit multiplier in pokemon games. Critical hits have a random 1/24 chance of occurring, and multiply the damage by 1.5 when occurring. Random is the mechanic that gives small variations to moves, known as rolls. Random is a random floating number ranging from 0.85 to 1. This gives moves a possible range of 16 slightly different values. STAB, or same type attack bonus, is a multiplier of 1.5 that occurs when a pokemon uses a move that is of the same type as its own type(s). The **damage_calc** function uses this formula, using a random value for the damage roll and the 1/24 chance for a critical hit, and a STAB multiplier. The function returns the raw damage done, as a float, which is then truncated in the **move_used** function.

Function move_order: The **move_order** function is called every iteration of the **turn** function. The arguments it takes are the two pokemon against each other each turn. It then compares the speed of both of them, returning which one is faster. If they are tied in speed, then the function returns "tie", which the **turn** function is able to randomly pick, when received.

Functions getdata and getmove_data: the functions **getdata** and **getmove_data** perform nearly identical tasks: They take a string as an argument, in the form of a title of a colum in the data spreadsheets. This, when used with Pandas' read_csv feature, allows us to easily call these functions multiple times to get the lists of each column from both the "updatedpokemon.csv" file, and the "moves.csv" file.

Function accuracy_check: the **accuracy_check** function is called every iteration of the **move_used** function. It takes a single integer as an argument: the move's accuracy. It runs a simple random test where the integer is compared between a random integer between 1 and 100. The function returns a string of either "hit" or "miss", which the **move_used** function uses to display and determine damage properly.

Function raw_stat: the **raw_stat** function takes a list of base stats, and converts them into raw numbers that can be used by the program's damage calculator. This is necessary, as the main games perform this calculation as well. There is a simple conversion from base stat to raw number, where the base stat is doubled, then 5 is added. There is a separate formula for converting base HP to raw HP, so it is not used with this function. The **raw_stat** function returns the corrected list for the stat.

<u>Function switch</u>: the **switch** function is called in the **turn** function whenever the player chooses to switch out their pokemon, and bring in a different one. The arguments it takes are the player's team, the pokemon they are switching out, and which player is switching. The function displays the player's team, including all of their HP percentages. It prompts the player to enter a valid entry for their pokemon to switch in, then returns the index of the player's team for the **turn** function to process correctly. It utilizes exception handling to prevent the user from entering a value besides the indexes for the user's team, and to prevent the user from attempting to swap in the same pokemon that they are attempting to swap out.

<u>Function main_loop</u>: the **main_loop** function is called at the very end of the **turn** function. The arguments it takes are both teams, both of the current pokemon during the turn, the turn count, a boolean operator representing if a pokemon was knocked out. The **main_loop** function first checks if the KO check is false. If it is false, it calls the **turn** function, continuing the battle. If it is true, it finds out which team the knocked out pokemon belongs to, and it deletes that entire entry from the team dictionary. Next, it checks to see if the length of the team is zero. If this is true, it displays a message saying who won the battle, and the function returns, ending the program. If the length of the team is not zero, the battle needs to continue by calling the **turn** function. But, a new pokemon must be sent in, as one has been knocked out. This is where the **switch_in** function is used, not to be confused with the **switch** function. Because the **switch_in** function returns the index of the pokemon the player chooses to switch in, and the **turn** function takes the pokemon currently on the field as part of its arguments, the call for the **switch_in** function can be that argument for the calling of the **turn** function. For example, if a pokemon on player 2's team is knocked out, it will first be deleted from the team dictionary, then the **turn** function will be called to continue the battle. But, since that pokemon has been deleted, the **turn** function needs player 2's current pokemon to take as one of its arguments. Calling the **switch_in**

function returns the index of the new pokemon as player 2's current pokemon. This can be used with dictionary comprehension to get the pokemon itself, allowing the battle to continue with the new pokemon on the field.

Function switch_in: Lastly, the **switch_in** function is called in the **main_loop** function whenever a pokemon is knocked out, as a new one needs to be switched in. The arguments it takes are the player number, and that player's team. The function first displays the player's whole team and their HP percentages. Then, it prompts the user to enter a valid index for a new pokemon to switch in, using exception handling to accept a valid input. It displays a message telling that the player sent out the new pokemon, then returning that pokemon's index for the **main_loop** function to use.

PROGRAM WEAKNESSES

The main weakness of this program is the fact that it was not approached from an object-oriented perspective. It would likely be significantly more organized and easier to debug if instead of multiple lists and dictionaries containing pokemon data, there was class for pokemon, with each object containing all the different attributes for each pokemon. Several methods would allow for simpler and quicker initialization. Additionally, further utilization of Pandas using dataframes would allow for pokemon and move data to be more organized, after extracting them from the spreadsheets.

ADDITIONAL CONTRIBUTIONS

It is likely that I will continue to work on this project in the future, and I will likely modify it to be object-oriented. Even after this large change, there are several features that I did not implement in my battle simulator that I would like to. The main feature I want to add is a graphical user interface for players, allowing them to see their pokemon, team, HP, moves, etc. Additionally, this would allow for the program to be played realistically between two players, as player choices would be hidden from the opposing player. This is how the Pokemon games are normally played, where players are on two different devices in the same battle, unable to see what choice a player makes. Additionally, there are several other key features in pokemon battles that are absent in this simulator. This includes additional effects from moves, priority moves, moves with a high critical hit chance, abilities, items, different pokemon forms, and pokemon abilities. All of these features would take very long to implement, and were not possible or feasible for this project given the due date. Lastly, I would like to implement more features that I have learned in this course, such as possibly visualizing battle data and pokemon data using Matplotlib. All of these features will likely implemented into this project in the future.

CONCLUSION

The goal of this project was to create a functioning program that simulates a Pokemon battle in Python. This was accomplished by importing and processing game data into data structures, creating teams for both players, and utilizing functions created to simulate battle features. In order to best understand the battle process and the program’s functions, an introduction to Pokemon and its features was required. This project utilized many of the concepts learned throughout this course. This includes functions, conditionals, looping, string formatting, list and dictionary comprehension and indexing, file reading and processing, and several libraries including math, random, NumPy, and Pandas. Additionally, the project enabled me to learn a significant amount about Python, and many of its features not fully covered throughout this course. This included exception handling, the JSON library, and the time library. Because this project was entirely my own work minus the data that I imported for the game to process, I gained a very strong knowledge of Python as a programming language and many of its key features. Based on this, it can be concluded that choosing this topic as my final project was very beneficial for anything I choose to make in Python in the future.