CMP7213 – Modern Optimisation

Pokémon battles and game theory

Coursework Assignment Report

Zan Zver
18133498

Word count
2850

MSc Big Data Analytics
Faculty of Computing, Engineering and the Built Environment
Birmingham City University

Contents

List of Tables

List of Figures

Abstract

Goal of this report is to simulate a battle between a bot (that has randomly chosen Pokémon) and Genetic Algorithms (that filters its Pokémon).

Choosing the Pokémon for battle is hard since there are always advantages and disadvantages. If Genetic Algorithms can eliminate the process and chose the best Pokémon for us, the task would be marked as successful.

In the report, there are different methods that were tried in order to see what improves fitness score. It turns out that elitism was the biggest improvement and gambling payed off for one of the examples.

1 Introduction

The report focuses on use of Genetic Algorithms (GA) with Pokémon battles. Before the battle, Pokémon has to be chosen. If it is 3v3 (Player Vs Player), each of the players needs to choose 3 Pokémon that they can use in the battle. Other option is Player Vs Environment (PVE) where there is 6 Pokémon that player chooses and there is 1 boss (much stronger) Pokémon that the player fights.

Key problem is in choosing correct Pokémon, either for PVP or PVE. This falls under game theory, were (computer) algorithm(s) must make decisions as humans would (in choosing the best set of Pokémon).

GA is used to tackle this problem due to its ability to tests different scenarios and evolve. Idea is for opponent (bot) to have 3 random Pokémon and our GA to find the best opponents. Algorithm would start by choosing random 3 Pokémon, comparing score against bot and switching Pokémon in the set until the best set is found.

2 Problem domain

Looking at the problem from wider angle, it can be marked as game theory. Deeper dive is in Section 3.1.

Implementation Pokémon battles is a multi stage process. Each Pokémon battle consist of:

1. Selecting the team
   - Each player (2 players total) has to select n (3 in our case) Pokémon. They have to chose from 1025 available Pokémon. Creating a balance is important, since every Pokémon as its strengths and weaknesses. Another thing worth considering is Pokémon types. For example, water type Pokémon will deal 2x the damage on fire type Pokémon.
2. Turn-based actions
   - Once in the battle, player has option to attack opponents Pokémon or switch to different Pokémon. Only one action can be made per turn, meaning it is rotated between players.
3. Moves and Effects
   - Once player attacks opponents Pokémon, it has to be mindful of moves and effects his and opponents Pokémon have. Move can deal 2x the damage based on Pokémon type. There can be side effects such as sleep where turn is skipped due to it.
4. Winning
   - First player to defeat all 3 opponent's Pokémon wins the battle. Alternatively, any player who forfeits looses. There are no draws, therefore one winner is guaranteed.

3 Problem instance

In order to get a deeper understanding of the task, we can break it down. First point is the problem we are trying to solve and second is the dataset that is going to be used.

3.1 Problem

Problem with Pokémon battles is choosing the best set in order to defeat opponent(s) Pokémon. This can be linked to game theory.

Core of game theory is decision making (Gibbons and Gibbons 1992). Based on actions of someone else (e.g. opponent) reaction that is optimal to us is chosen. This can all be described as decision trees (Fudenberg and Tirole 1991) and improved with genetic algorithms (Périaux et al. 2001) or with modern solutions such as AI (Kusyk et al. 2021).

Objective of the main problem can be described "creation of the most efficient team". Result would be maximising opportunity to win, due to having the best (most optimal) set of Pokémon(s) against opponent.

How to overcome the problem (as PVP) can be seen in flowchart (Figure 1). It has 4 main steps:

- Step 1
  - 3 random Pokémon are generated as p1, p2 and p3.
- Step 2
  - Counter Pokémons are created, resulting in 3 datasets.
- Step 3
  - Objective function is applied to each counter (c1, c2, c3) in order to find the best Pokémon. This is done by trying to find the best attack to HP ratio.
- Step 4
  - Team of Pokémon is assembled based on top Pokémons in c1, c2 and c3.

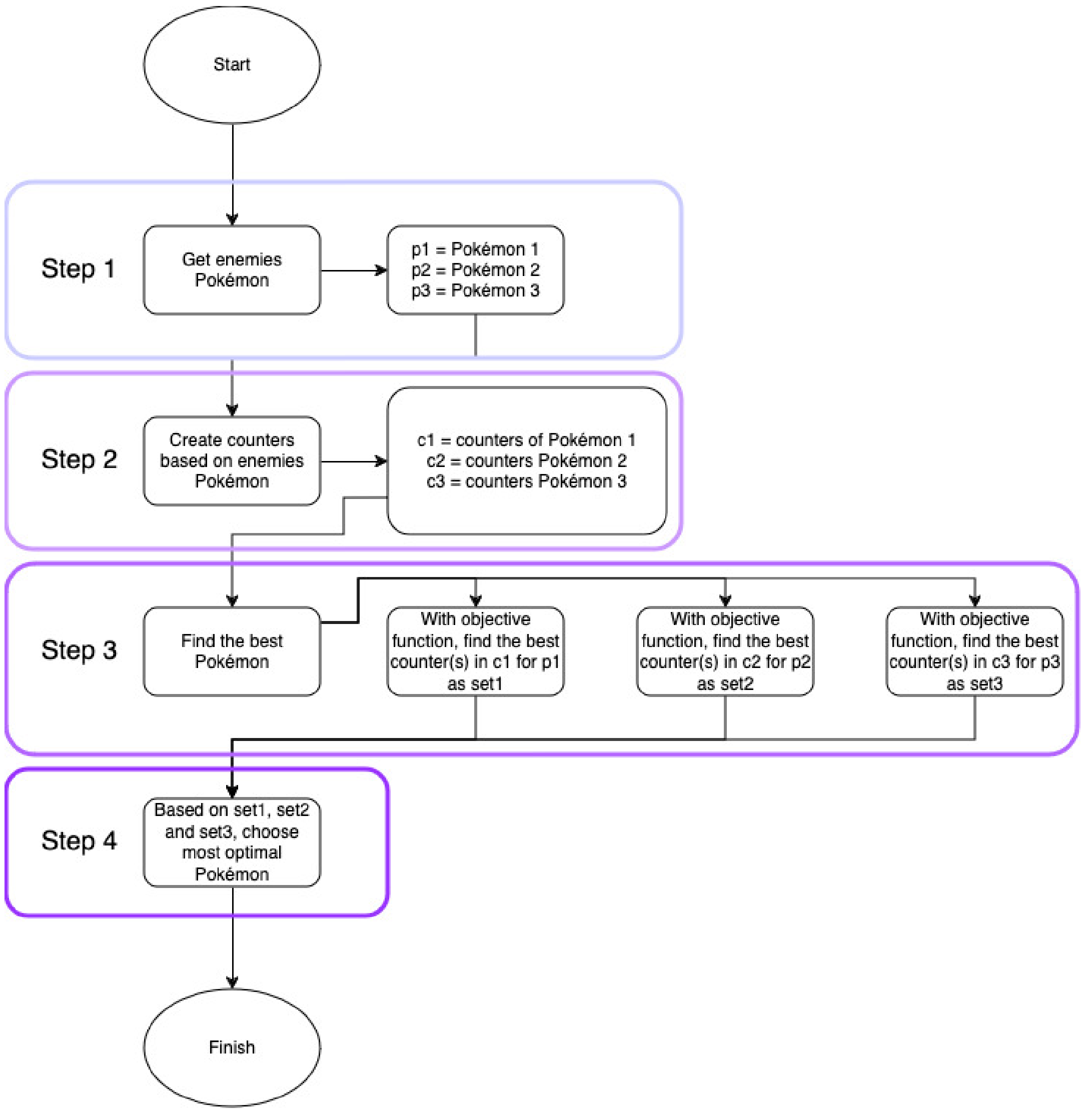

Start
Step 1
Get enemies Pokémon
p1 = Pokémon 1
p2 = Pokémon 2
p3 = Pokémon 3
Step 2
Create counters based on enemies Pokémon
c1 = counters of Pokémon 1
c2 = counters Pokémon 2
c3 = counters Pokémon 3
Step 3
Find the best Pokémon
With objective function, find the best counter(s) in c1 for p1 as set1
With objective function, find the best counter(s) in c2 for p2 as set2
With objective function, find the best counter(s) in c3 for p3 as set3
Step 4
Based on set1, set2 and set3, choose most optimal Pokémon
Finish

Figure 1: Execution plan for PVP

Due to two battle options (PVP and PVE), program has to know what is recommending. This would result in having two algorithms (at the end, only PVP was implemented). One for PVP (player vs player) and one for PVE (player vs environment). How PVP and PVE work can be seen in 9.1.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

Equation 1: Formula for Pokémon battles

Meaning:

- $n!$ represents the factorial of n, which is the product of all positive integers less than or equal to n. In our case, total number of Pokémon available to choose from (3 per player, 6 in total in an instance of PVP).

- $k!$ represents the factorial of k. This would be number of Pokémon each player has per battle (3 for PVP).
- $(n-k)!$ represents the factorial of $n-k$. This calculates number of remaining Pokémon to select from.

This can also be represented as:

$$\binom{6}{3} \times \binom{6}{3} \tag{2}$$

Equation 2: Representation of PVP expression

Each Pokémon is of different type. When this is taken into the consideration, we can create the formula for calculating the best Pokémon as:

$$\text{attack} \times (\text{impact from type 1}) + \text{defense} \times (\text{impact from type 1}) = \text{total power}$$

Calculation for best Pokémon

Assume opponent has fire type Pokémon. We have Pokémon power of 30 and defense of 70. Calculations in Figure 11 show the impact of Pokémon types.

$$30 \times 2 + 70 \times 2 = 60 + 140 = 200 \tag{3}$$

Equation 3: Example with water Pokémon

$$30 \times 0.5 + 70 \times 1 = 15 + 70 = 85 \tag{4}$$

Equation 4: Example with bug Pokémon

Goal is to maximise the sum, meaning Pokémon with sum of 200 would be better (due to water beating fire). Calculation like that (without the sum) is applied to whole dataset in order for algorithms to find the optimal solutions between attack and defence.

3.2 Dataset

Data used in the report is from Pokebase (3rd party Python library) (Pokebase n.d.). Underneath, they are translating interactions from PokeAPI (PokeAPI n.d.).

First iteration of the code used sequential calling of the API. This means that API was called every generation, thus rate limiting and slowing down the operation. By chancing this to batch (downloading once), file is stored locally meaning there aren't any rate limits or networking latencies involved. This resulted in 2h+ task to be completed under 2min. Dataset is included in GitHub (Zver n.d.), therefore it doesn't need to be downloaded.

Once data is loaded, it represents 1025 Pokémon. Example of first Pokémon can be seen in figure Figure 2.

```
        "bulbasaur": {
            "name": "Bulbasaur",
            "stats": {
                "hp": 45,
                "attack": 49,
                "defense": 49,
                "special-attack": 65,
                "special-defense": 65,
                "speed": 45
            }
        },
```

Figure 2: JSON representation of Bulbasaur's attributes

Analyzing the dataset further down, constraints can be set as JSON schema (seen in Figure 1). This would guarantee for data to be ingested in desired format. JSON schema is usually used in cases when data is change, but in case of Pokémon it is static.

Amazon Web Services (AWS) did a write up on JSON schema (Services n.d.). Their example was pet store where there were allowed pets, price ranges and data type limits. When someone would call their (pet store) API, we would know not to expect whale as an animal with price of 50k (since JSON schema would flag it).

```
{
  "schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "bulbasaur": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "stats": {
          "type": "object",
          "properties": {
            "hp": {
              "type": "integer",
              "minimum": 0
            },
            "attack": {
              "type": "integer",
              "minimum": 0
            },
            "defense": {
              "type": "integer",
              "minimum": 0
            },
            "special-attack": {
              "type": "integer",
              "minimum": 0
            },
            "special-defense": {
              "type": "integer",
              "minimum": 0
            },
            "speed": {
              "type": "integer",
              "minimum": 0
            }
          },
          "required": ["hp", "attack", "defense", "special-attack", "special-defense", "speed"]
        }
      },
      "required": ["name", "stats"]
    }
  },
  "required": ["bulbasaur"]
}
```

Source Code 1: JSON Schema for validating Pokémon data

There are 7 attributes each Pokémon has and they are described in the Table 1. All of the attributes (except name of the Pokémon) are of type int. The number must be at minimum 0 (aka non-negative integer). Pokémon names (of type string) must be unique while other values don't need to be.

3.3 Chromosome

Each of the Pokémon has different attributes as described in data section (Section 3.2). In 3v3 (PVP), there are two sets of Pokémon, one that enemy has and ours. In a set, there are 3 Pokémon (example set shown in Table 2). Goal is to change Pokémon around in order to get the best combination of attributes.

Attribute	**Description**
Name	Name of the Pokémon.
HP	Hhealth points of Pokémon which determine how much damage a Pokémon can take before fainting.
Attack	Measure of how powerful the Pokémon's physical attacks are.
Defense	Ability of the Pokémon to resist physical attacks.
Special Attack	Measure of the power of the Pokémon's special moves.
Special Defense	Pokémon's ability to withstand special moves from opponents.
Speed	Determines how quickly the Pokémon can act in battle, influencing the order of taking turns.

Table 1: Pokémon Data Attributes

Pokémon	**HP**	**Speed**	**Defense**	**Attack**	**Special Attack**	**Special Defense**
Pikachu	35	90	40	55	50	50
Charizard	78	100	78	84	109	85
Bulbasaur	45	45	49	49	65	65

Table 2: Example of data input for a Chromosome

Once set is created, we extract chromosome from it. This is done by sum of attributes (seen in Equation 5).

The sum of attributes S for a Pokémon is given by:

$$S = HP + Spd + Def + Atk + SpA + SpD \tag{5}$$

Equation 5: Total sum of attributes for the Pokémon

Where:

- HP is the Health Points,
- Spd is the Speed,
- Def is the Defense,
- Atk is the Attack,

- SpA is the Special Attack,
- SpD is the Special Defense.

For example, the sum of attributes for Pikachu would be calculated as:

$$S_{\text{Pikachu}} = 35 + 90 + 40 + 55 + 50 + 50 = 320 \tag{6}$$

Equation 6: Total sum of attributes for Pikachu

The total sum T of the attribute scores for all Pokémon is calculated as:

$$T = S_{\text{Pokémon 1}} + S_{\text{Pokémon 2}} + S_{\text{Pokémon 3}} \tag{7}$$

Equation 7: Total sum of attributes for all Pokémon

Looking at the Pokémon from the example (Table 2), this can be represented as:

$$T = S_{\text{Pikachu}} + S_{\text{Charizard}} + S_{\text{Bulbasaur}} \tag{8}$$

Equation 8: Total sum of attributes for all Pokémon

Where:

- $S_{\text{Pikachu}} = 320$,
- $S_{\text{Charizard}} = 534$,
- $S_{\text{Bulbasaur}} = 318$.

Therefore, the calculation is:

$$T = 320 + 534 + 318 = 1172 \tag{9}$$

Equation 9: Calculation of the total sum

T represents total score of each set. Goal is to maximise T. This is done by shuffling Pokémon around in order to see which ones would generate the biggest T.

After all the calculations, we can see our chromosome in Table 3.

3.4 Fitness function

The chromosome section (Section 3.3), mentions how chromosome is created and what it represents. Total (T) needs to be higher compared to opponents total in order for our set to win. This implies that maximisation of T is required.

In order to make the calculation, our set and opponents set are passed in. Afterwards, we get stats from the dataset and calculate which set has higher score.

$$f(I, O) = \sum_{p_i \in I} \sum_{p_j \in O} \sum_{s \in S_{p_i}} (s(p_i) > s(p_j)) \tag{10}$$

Pokémon	**Sum**
Pikachu	320
Charizard	534
Bulbasaur	318
Total	**1172**

Table 3: Created GA Chromosome example

Equation 10: This function calculates the sum of scores where $s(p_i) > s(p_j)$ for all $p_i \in I$ and $p_j \in O$.

- I is the individual team, consisting of Pokémon p_i,
- O is the opponent team, consisting of Pokémon p_j,
- S_{p_i} represents the set of stats for a Pokémon p_i,
- $s(p_i)$ and $s(p_j)$ are the values of stat s for Pokémon p_i and p_j respectively,
- $(s(p_i) > s(p_j))$ is an indicator function that evaluates to 1 if $s(p_i)$ is greater than $s(p_j)$, and 0 otherwise.

Using code, this can be represented as:

```python
def eval_team(individual, opponent_team):
    score = 0
    for Pokémon in individual:
        for opponent in opponent_team:
            score += sum(
                Pokémon['stats'][stat] > opponent['stats'][stat]
                for stat in Pokémon['stats']
            )
    return (score,)
```

Source Code 2: Fitness function calculation

4 Candidate optimisation methods

Paper by (Cole, Louis, and Miles 2004) has used GA in video game. The parameters used in this report are the same as parameters in their paper since topic is the same (video game). Parameters can be seen in Table 5.

Not all the parameters between papers are the same (e.g. chromosome length, selection strategy). Therefore an approximation had to be done in order to choose matching parameters.

4.1 Method 1: Use elitism

Use of elitism is something that has been talked in the past (De Jong 1975). What it does, is it makes sure the best candidates from each generations are carried over. This prevents algorithm loosing best solutions (Ahn and Ramakrishna 2003).

In our example, if we had the best Pokémon set in generation 7 (as example), with use of elitism this will be carried thought generations. If there isn't any better sets at the end, the set from generation 7 would win.

4.2 Method 2: Change selection strategy

Authors in original paper (Cole, Louis, and Miles 2004) have used tournament selection. There are other strategies that can work better or maybe worse.

4.2.1 Tournament selection

This is most known selection strategy. A set of individuals is chosen (randomly) from the population and the best ones are carried (able to reproduce) (Miller, Goldberg, et al. 1995).

E.g.: Out of 1025 Pokémon, N number is randomly chosen. Out of that, the best Pokémon can go into the next stage.

4.2.2 Roulette selection

With roulette, every individual is assigned a space on the roulette wheel. How big the space is depends on fitness of said individual (making it unfair wheel) (Yu et al. 2016). When spinning the wheel, chances of selecting better individuals are greater.

E.g.: 1025 Pokémon are mapped on the wheel. When we spin it, the chances of us getting better Pokémon are higher.

4.2.3 Rank selection

Rank strategy, involves ranking (based on fitness) all of the individuals in the population (Goldberg and Deb 1991). Instead of selecting based on fitness all the individuals are ranked based on given rank.

E.g.: Fitness is calculated for Pokémon. Out of 1025 Pokémon, we rank them based on the given fitness.

4.3 Method 3: Increase number of generations

By increasing number of generations, we allow GA to run longer and therefore produce better results (Vrajitoru 2000). This does mean increased computational cost, longer run time and overfitting. By increasing the number, it is also shown if increase was worth it or if optimal solution was found early (therefore not needing the increase).

E.g.: When in search for most optimal Pokémon set, we end at 30 tries. By expanding that to 50, we can see if extra 20 tries has yielded in any better results or if we have gotten best set by 30th generation.

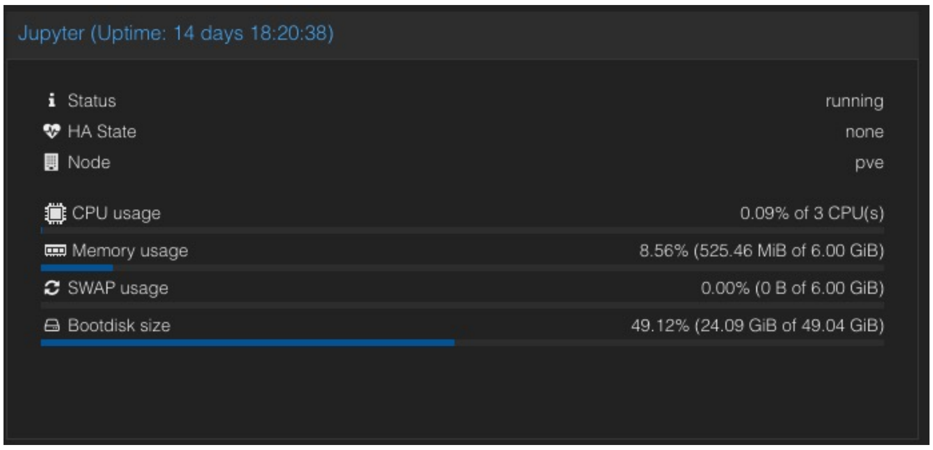
5 Experimental setup

Developed program accepts 7 different parameters as an input. Different optimisation methods are created by changing (optimising) one of the parameters at the time. Description of parameters is represented in Table 4.

Parameter	**Description**	**Valid Values**
selection_strategy	Method used for selecting individuals.	tournament or roulette or rank
tournament_size	Number of individuals competing in each tournament.	Positive integer. Only used if selection_strategy value is tournament.
use_elitism	Whether to use elitism to carry the best individuals to the next generation.	True or False
number_of_generations	Total number of generations for GA execution.	Positive integer
population_size	Number of individuals in the population.	Positive integer
cxpb	Crossover probability between two individuals.	Floating-point number between 0 and 1
mutpb	Mutation probability of an individual's gene.	Floating-point number between 0 and 1

Table 4: Accepted input parameters and their explanation

Tests were executed in the virtual (isolated) environment (seen in Figure 3). Recommended setup is at least 3 core CPU and 6GB of memory (RAM).



Jupyter (Uptime: 14 days 18:20:38)

Status	running
HA State	none
Node	pve
CPU usage	0.09% of 3 CPU(s)
Memory usage	8.56% (525.46 MiB of 6.00 GiB)
SWAP usage	0.00% (0 B of 6.00 GiB)
Bootdisk size	49.12% (24.09 GiB of 49.04 GiB)

Figure 3: VM setup on Proxmox

Virtual machine (VM) is not needed to reproduce the results, it can be done on any machine. This has been described in step by step guide on GitHub (Zver n.d.).

6 Results

This section provides results of methods described in Section 4. Each test was ran multiple times. This can be seen in the legend when different runs of tests are represented (GA1, GA2 and GA3).

Tests can be reproduced by manually entering parameters or by navigating to GitHub (Zver n.d.) and using provided code from there.

6.1 Baseline

First test establishes baseline. Parameters are the same as in original article (seen in Table 5). Fitness score is in low 50s and this was found after 21 generations (Figure 4). Although the data is different, results seem to be acceptable. It is not transparent from original paper if units for fitness are multiplied (100x) in the code or not. In case they are, results shown are similar (a bit better), otherwise they are 100x worse.

Parameter	**Value**
selection_strategy	tournament
tournament_size	5
use_elitism	False
number_of_generations	50
population_size	10
cxpb	0.95
mutpb	0.1

Table 5: Baseline

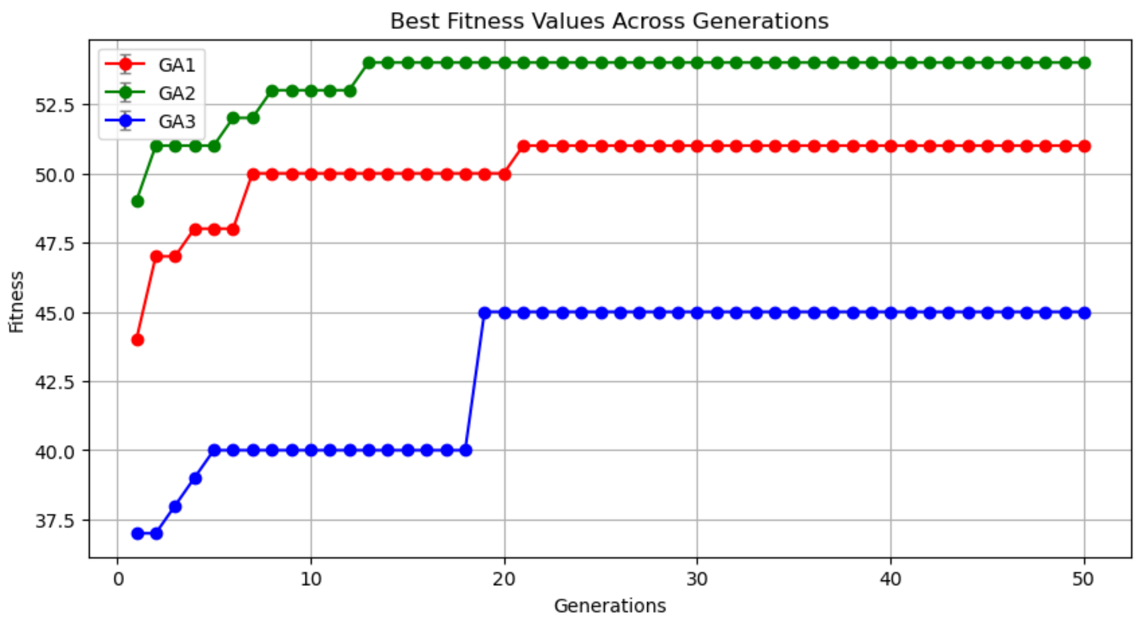


Figure 4: Baseline

6.2 Method 1: Use of elitism

When using elitism, what can happen is that in fist generation the most optimal candidate is found (Ahn and Ramakrishna 2003). This happened in GA3 (Figure 5), resulting in a flat line (parameters found in Table 6). Other two tests needed at least 13 generations in order to get to the best score. Average fitness score was in mid 50s.

Parameter	**Value**
selection_strategy	tournament
tournament_size	5
use_elitism	True
number_of_generations	50
population_size	10
cxpb	0.95
mutpb	0.1

Table 6: Method one elitism parameters

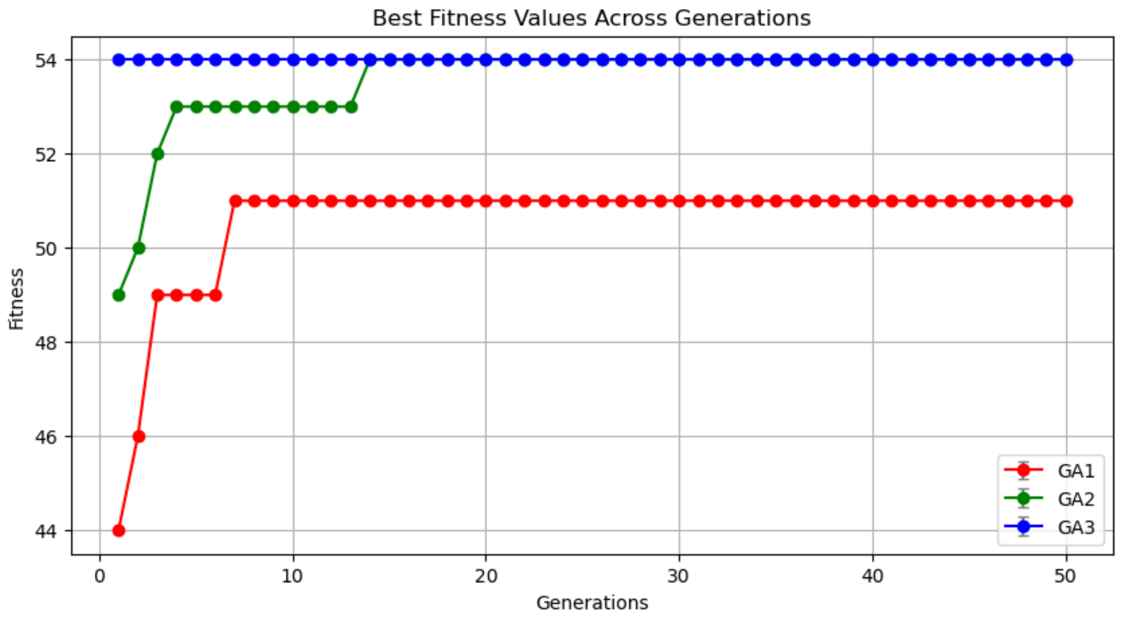


Figure 5: Method one elitism results

6.3 Method 2: Change selection strategy

Tournament selection can be seen in baseline Section 6.1, therefore the test will not be repeated.

6.3.1 Rank selection

Rank selection was all over the place, this is one of the pitfalls of the algorithm (Shukla, Pandey, and Mehrotra 2015). GA3 was somewhat consistent (Figure 6, indicating that maybe with a few more generations it an improve. GA1 and GA2 were not as consistent on the other hand (although having same parameters Table 7. Looking at the fitness score, GA3 went from 47 to 51, 4 points of change (consistent). GA1 and GA2 both started low at 35 and never gotten as high.

Parameter	**Value**
selection_strategy	rank
tournament_size	NA
use_elitism	False
number_of_generations	50
population_size	10
cxpb	0.95
mutpb	0.1

Table 7: Method two rank parameters

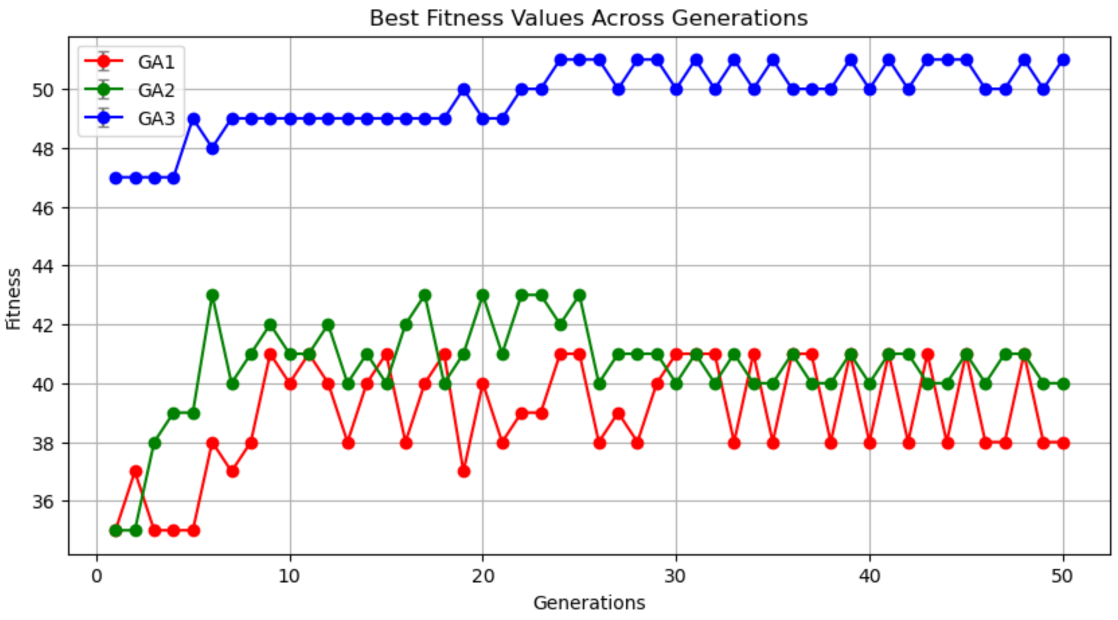


Figure 6: Method two rank results

6.3.2 Roulette selection

Roulette presented much more balanced approach (Yu et al. 2016). Alto all tests (GA1, GA2 and GA3) provide different results (Figure 7) with same parameters (Table 8, they are all more balanced. After 10 generations, we wouldn't need any more exploring due to marginal improvements.

Parameter	**Value**
selection_strategy	roulette
tournament_size	NA
use_elitism	False
number_of_generations	50
population_size	10
cxpb	0.95
mutpb	0.1

Table 8: Method two roulette parameters

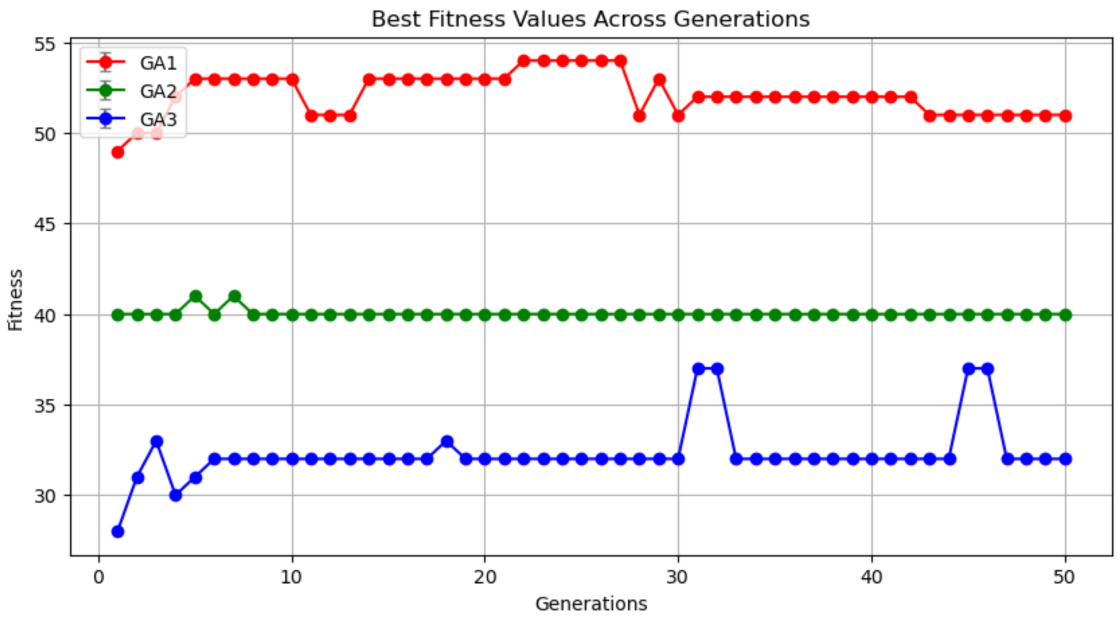


Figure 7: Method two roulette results

6.4 Method 3: Increase number of generations

When allowing algorithm to work longer and explore other options, there could be different outcomes (Gibbs et al. 2006). Not in this case. When doubling number of generations (from 50 to 100, Table 10), result didn't change (Figure 8). After generation 26, there wasn't much of an improvement.

Parameter	**Value**
selection_strategy	tournament
tournament_size	5
use_elitism	False
number_of_generations	100
population_size	10
cxpb	0.95
mutpb	0.1

Table 9: Baseline

Table 10: Method three generation increase parameters

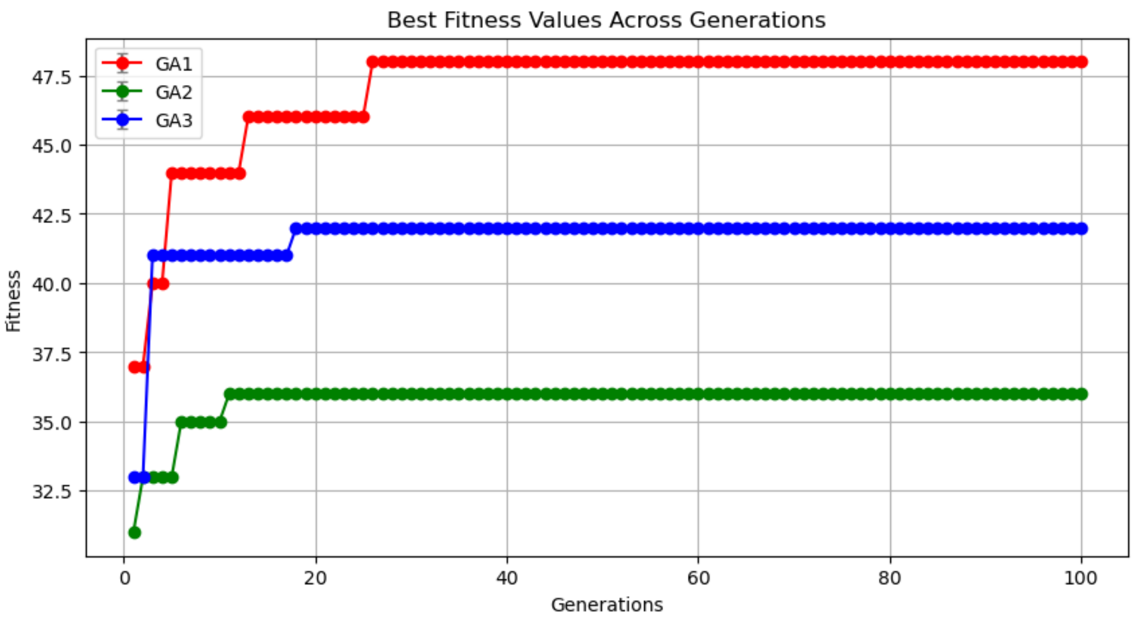


Figure 8: Method three generation increase results

7 Discussion

Results section (Section 6) displays 5 different graphs. Expectations for elitism were high due to its use cases. It does typically perform well, due to keeping best individuals.

What was surprising was roulette. It did perform well, expectations for it were lower due to gambling approach (random). It can perform great (shown in GA1) and poor (GA3) (Figure 7.

Increasing generation(s) did result in nothing. This is somewhat surprising. Most often, by this could be due to lack of diversity. To fix this issue, further parameter tuning would be required. By chaining crossover (cxpb) and mutation (mutpb), genes could react differently.

8 Conclusion

Use of GA in Pokémon battles was the task this report tried to solve and it did it. Mimicking enemy player and GA player, battles were simulated in order to test different methods of GA.

Unsurprisingly, using elitism increased our score. Possibly due to lack of diversity, increasing generations did not help us. What can be seen is how roulette approach has 3 completely different versions due to gambling.

For future work, it would be interesting to see if score can be improved. Another thing would be integration with PvPoke (PvPoke n.d.). This would allow GA to battle real players instead of bots.

In the report, it was mentioned PVP and PVE battles. So far, only PVP was implemented. Adding a flag (as additional parameter) to indicate what kind of battle it is would be an improvement. Comparison could be drawn if an algorithm does act different in PVP and PVE mode. In case there would be difference, new GA would probably be needed.

The data that is used in battles includes legendary Pokémon. While this doesn't represent a problem (since they are allowed in battles), it could change the scales. Legendary Pokémon usually has better stats (since it is legendary). It would be interesting to remove them from equation in order to see if having them in skew results (aka GA always chooses legendary Pokémons).

9 Appendix

9.1 Appendix 1

First battle style is PVE (also know as raids) (Fandom n.d.[a]). In this case, we have a boss Pokémon (that is stronger) and 6 Pokémons available for battle. Formula for Pokémon battles is shown in Equation 11. Figure 9 showcases battle example.

$$\binom{6}{6} \tag{11}$$

Equation 11: Representation of PVE expression

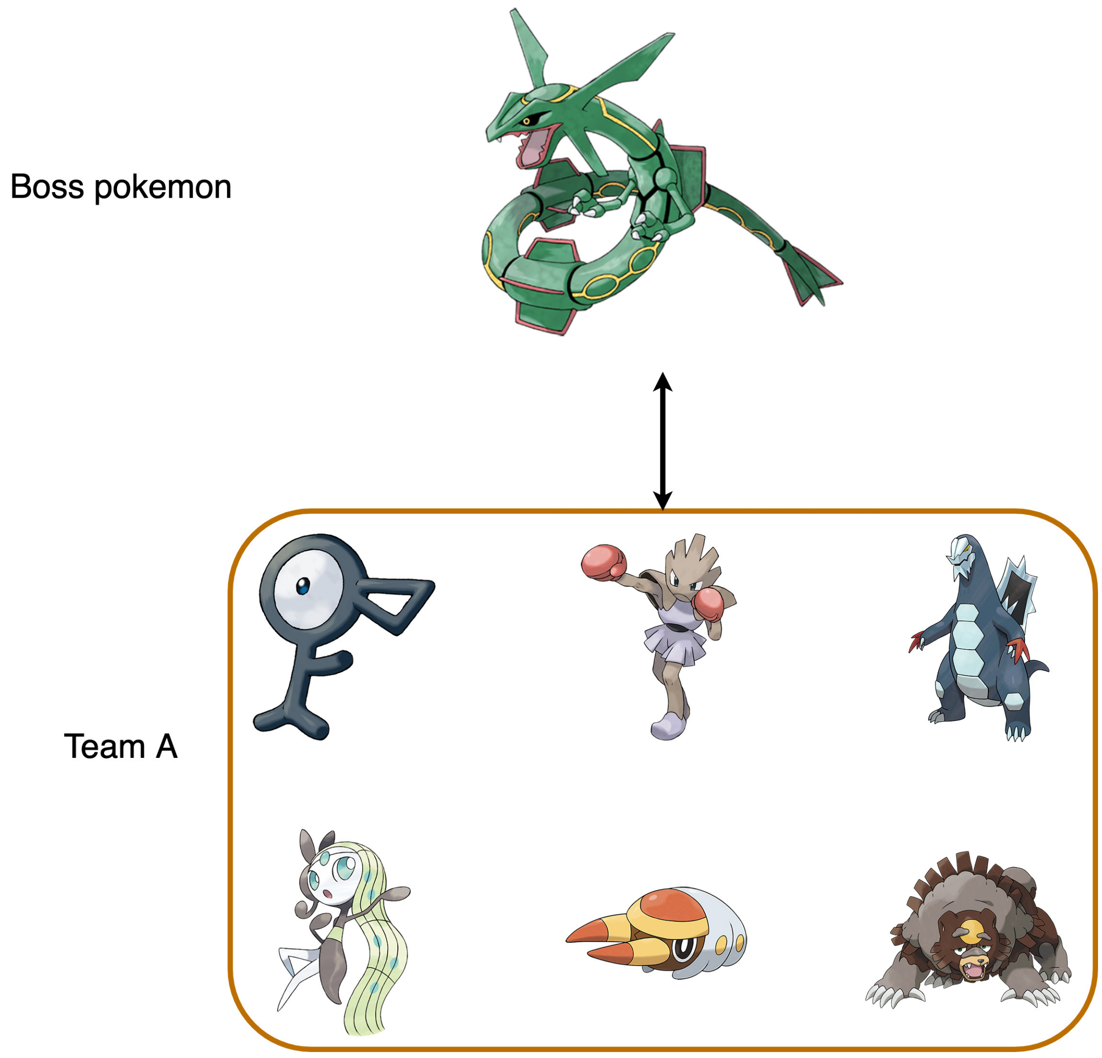


Figure 9: PVE example

In PVP battles (Fandom n.d.[b]), there would be 3v3 Pokémons (enemy vs yourself), meaning winner will use at least one Pokémon and other participant would have to use all 3 of their Pokémon. We can represent this as shown in Equation 1 for Pokémon battles.

Expression for PVP is a bit different. Essentially there is 6 Pokémon all together but you can only have 3 (each) in the battle and once Pokémon is defeated, it cannot battle anymore.

Example of the PVP battle can be seen in Figure 10.

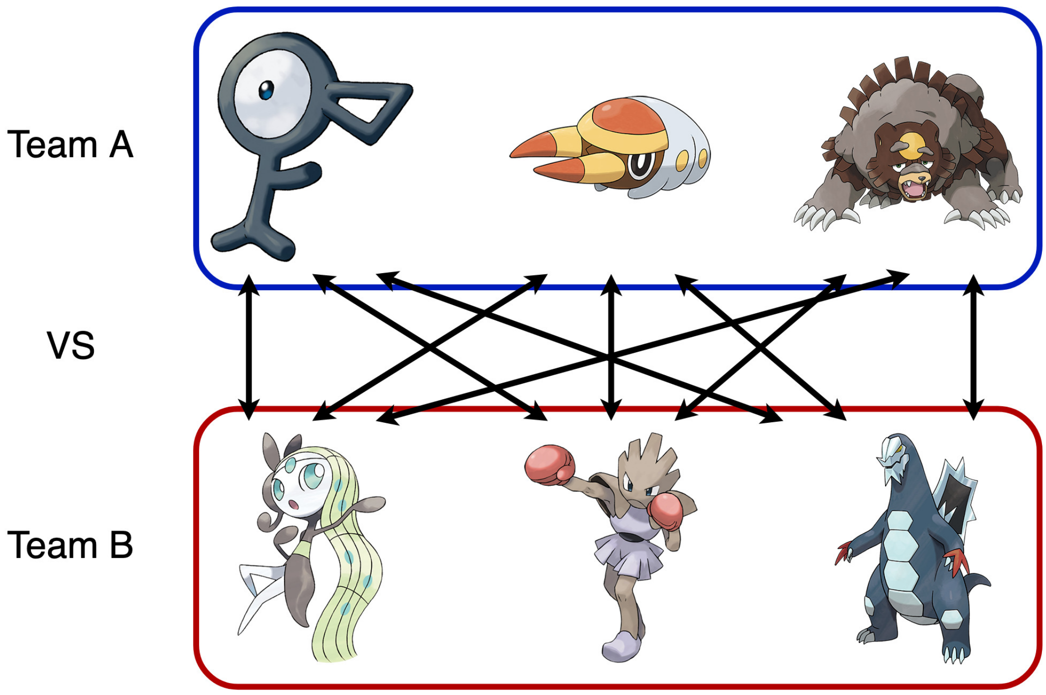
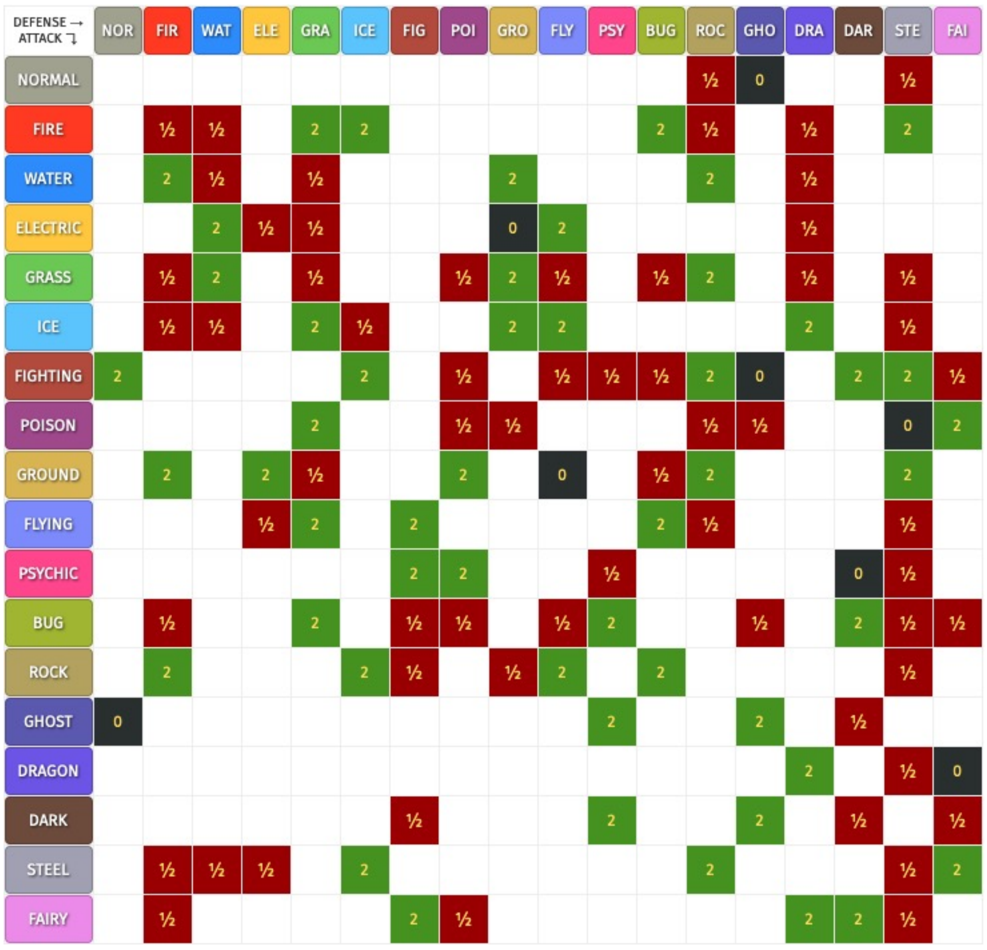


Figure 10: PVP example

What we need to take into consideration is Pokemon types (PokemonDB n.d.). As Figure 11 shows, some Pokemon types deal 2x damage, 0.5 or none.



DEFENSE → ATTACK ⤵	NOR	FIR	WAT	ELE	GRA	ICE	FIG	POI	GRO	FLY	PSY	BUG	ROC	GHO	DRA	DAR	STE	FAI
NORMAL													½	0			½	
FIRE		½	½		2	2						2	½		½		2	
WATER		2	½		½				2				2		½			
ELECTRIC			2	½	½				0	2					½			
GRASS		½	2		½			½	2	½		½	2		½		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		2	½			2		0		½	2				2	
FLYING				½	2		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	

Figure 11: Counter types

10 References

References

Ahn, Chang Wook and Rudrapatna S Ramakrishna (2003). "Elitism-based compact genetic algorithms". In: *IEEE Transactions on Evolutionary Computation* 7.4, pp. 367–385.

Cole, Nicholas, Sushil J Louis, and Chris Miles (2004). "Using a genetic algorithm to tune first-person shooter bots". In: *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753).* Vol. 1. IEEE, pp. 139–145.

De Jong, Kenneth Alan (1975). *An analysis of the behavior of a class of genetic adaptive systems.* University of Michigan.

Fandom, Pokemongo (n.d.[a]). *Raid Battle.* Last accessed 18 Feb 2024. URL: https://pokemongo.fandom.com/wiki/Raid_Battle.

— (n.d.[b]). *Trainer Battle.* Last accessed 18 Feb 2024. URL: https://pokemongo.fandom.com/wiki/Trainer_Battle.

Fudenberg, Drew and Jean Tirole (1991). *Game theory.* MIT press.

Gibbons, Robert and Robert Gibbons (1992). "A primer in game theory". In.

Gibbs, Matthew S et al. (2006). "Minimum number of generations required for convergence of genetic algorithms". In: *2006 IEEE International Conference on Evolutionary Computation.* IEEE, pp. 565–572.

Goldberg, David E and Kalyanmoy Deb (1991). "A comparative analysis of selection schemes used in genetic algorithms". In: *Foundations of genetic algorithms.* Vol. 1. Elsevier, pp. 69–93.

Kusyk, Janusz et al. (2021). "Artificial intelligence and game theory controlled autonomous UAV swarms". In: *Evolutionary Intelligence* 14, pp. 1775–1792.

Miller, Brad L, David E Goldberg, et al. (1995). "Genetic algorithms, tournament selection, and the effects of noise". In: *Complex systems* 9.3, pp. 193–212.

Périaux, Jacques et al. (2001). "Combining game theory and genetic algorithms with application to DDM-nozzle optimization problems". In: *Finite elements in analysis and design* 37.5, pp. 417–429.

PokeAPI (n.d.). *PokeAPI.* Last accessed 4 May 2024. URL: https://pokeapi.co/.

Pokebase (n.d.). *Pokebase.* Last accessed 4 May 2024. URL: https://github.com/PokeAPI/pokebase.

PokemonDB (n.d.). *PokemonType.* Last accessed 17 Mar 2024. URL: https://pokemondb.net/type.

PvPoke (n.d.). *PvPoke.* Last accessed 4 May 2024. URL: https://pvpoke.com/.

Services, Amazon Web (n.d.). *AWSJsonSchema.* Last accessed 4 May 2024. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/models-mappings-models.html.

Shukla, Anupriya, Hari Mohan Pandey, and Deepti Mehrotra (2015). "Comparative review of selection techniques in genetic algorithm". In: *2015 international* *conference on futuristic trends on computational analysis and knowledge management (ABLAZE)*. IEEE, pp. 515–519.

Vrajitoru, Dana (2000). “Large population or many generations for genetic algorithms? Implications in information retrieval”. In: *Soft computing in information retrieval: Techniques and applications*. Springer, pp. 199–222.

Yu, Fengrui et al. (2016). “Improved roulette wheel selection-based genetic algorithm for TSP”. In: *2016 International conference on network and information systems for computers (ICNISC)*. IEEE, pp. 151–154.

Zver, Zan (n.d.). *Source Code*. Last accessed 6 May 2024. URL: https://github.com/ZanZver/PokemonBattles.