Gotta Model ‘Em All! A Pokémon Agent-Based Model

Ali Turfah

CMPLXSYS 530

Introduction

There are three aspects of games that contribute significantly to their ability to be taken as competitive, namely the amount that strategy, randomness, and matchups play into the outcome of a game.

Everyone who approaches any game tends to play by some strategy, or guidelines to determine how they will play. These strategies can be, among others, predetermined, reactive, or random, and each has its own benefit given the situation. For example, against a reactive player it may be most beneficial to play erratically, whereas it is likely advantageous to play reactively to a very deterministic player.

Randomness and competitiveness are often at odds with one another—no one would consider winning a coin toss ‘skillful’. In a similar vein, many card games such as Hearthstone or Poker, while they do have aspects of strategy and matchup, can be won or lost very early on due to not being dealt the proper hand. Granted, randomness is not always a negative; ‘probability management’ is often considered part of a player’s ‘skill’ since it requires them to be able to adapt to situations outside of their control. The ability to rebound from unforeseen or uncontrollable situations is often a characteristic of a player with a deep understanding of their game. And ignoring the drive for ‘competitiveness’, random chance makes the game more enjoyable for a viewer, since the outcome of a game is not entirely deterministic.

Additionally, games that afford its players choices tend to suffer in some degree from ‘matchup’ problems; certain choices leave you weaker to certain opposing strategies. Rock-Paper-Scissors (RPS) epitomizes a game decided by matchup; paper always beats rock, rock always beats scissors, and scissors always beats paper. However, the cyclic matchup pattern assures that no single strategy dominates.

All games exist somewhere in a spectrum of randomness, strategy, and matchup. Chess, for example, has no built-in element of matchup or randomness; each player has the exact same pieces to work with and moves are purely deterministic. On the other extremes of randomness and matchup, we find coin tosses and RPS respectively. In the case of Pokémon, strategy, randomness, and matchup are all present. Strategy manifests itself in two places; in team construction as well as in the actual gameplay. Randomness plays a role almost every turn of a battle; move damage is based on a uniform distribution, secondary effects occur certain percent of the time, among other things. Lastly, certain Pokémon tend to do well against other Pokémon, due to how typing, statistic values, and moves are distributed (see Pokémon Terminology).

Purpose

The goals of this project are twofold: first, to create an accurate model of the Pokémon Showdown (PS) Simulator’s metagame, and secondly to model the actual battles as they occur in the simulator.

With regards to the metagame as a whole, the aspects of interest are which playing styles correspond to “high-level” play, as well as the extent to which the choice of Pokémon affects a strategy’s viability. In addition, the ability to project how ruleset changes, such as disallowing the use of a Pokémon, would affect the viability of other Pokémon is of interest.

With regards to the battles, the intention is to accurately model each turn of a battle, as agents using different strategies and Pokémon face off. This aspect of the simulation is what ultimately gives rise to the metagame level trends; however, it could also be used as a sandbox to allow newer players to get experience playing against a simulated opponent.

Problem Overview and Literature Review

ABM Justification

Mathematical or Verbal models would not be able to adequately represent this system. A mathematical model would just explode in the number of equations necessary to represent the system, which would not be able to capture the random aspects of the game. A verbal model, on the other hand, would have to be incredibly complex to account for the many types of situations that a game can take. Thus, agent-based modelling is an appealing choice because each player's teams and strategies can easily be represented as class attributes and functions respectively. It would also allow for a much finer way to incorporate subtle differences in how players make their moves with simple modifications to if-else statements.

Terminology

A Pokémon has one or two of 16 types, health, speed, defense and attacking values that are used to calculate how much damage it takes or receives, as well as how many attacks it can take before 'fainting'. If a Pokémon has 'fainted', it is unable to be used and therefore cannot be switched to. A battle is over when all of a player's Pokémon have fainted. Moves are either attacking or status moves, where the former's purpose is to deal damage whereas the latter is used for secondary effects like halving an opponent's speed or increasing the user's attack. In addition, each move can only be used a limited number of times, however this rarely comes into play. A set refers to a Pokémon with specific moves and battling statistic values. Variations on sets exist, where moves/statistics can be modified for specific uses. A team is a collection of one or more sets.

Elo rankings are used to estimate skill levels of players in games such as Chess (Glickman, 1995). Proposed by Arpad Elo in the 1970s, the Elo ranking system is used to rank players in games such as soccer, table tennis, chess, and Pokémon. While the exact calculations are a bit involved, it stands to know that a player with higher Elo ranking is expected to beat a player with a lower Elo ranking, and these odds can be calculated from their Elo scores.

Literature Review

There is very little literature on models of Pokémon metagames, as most real-time strategy analysis papers focus on StarCraft. Tavares et al.’s paper “Rock, Paper, StarCraft: Strategy Selection in Real-Time Strategy Games” details experiments matching five styles of StarCraft play, which served as an inspiration for the simulations used to evaluate this system. Cowling et al.’s “Search in Real-Time Video Games” details different algorithms used for searching in real-time video games, which helped inspire the strategy searching used in this project.

Entities

Agents

Agents in this model correspond to the game's players. All agents have a unique ID, a type (for logging purposes), and their Elo rating. For agents that play RPS, each agent has a strategy vector that corresponds to the probability they play one of Rock, Paper, or Scissors. An agent that plays Pokémon has a team that they use in the battles, as well as a method to calculate their next move. Finally, all agents have an internal game state, where they keep track of the relevant information they have gathered in this interaction.

Rules of Interaction

The rules of interaction are handled by the "Engine". These modules implement the rules of the game, including stopping conditions and turn-by-turn updating. The Engine is also responsible to inform the players of the results of a turn.

Interaction Topology

Players are matched by a "Ladder", which is a way to refer to the matchmaking service. The ladder has a pool of available players and chooses two to play based on its internal matchmaking function. For example, a ladder that matches players based on Elo rankings would match based on the difference in Elo rankings, whereas a random ladder would matched based on arbitrarily assigned values. Finally, the ladder updates the player's Elo rankings based on the results of the games.

Process Overview

The processes of this simulation occur in the Ladder, the Engine, and the Players.

Ladder Process

The Ladder, as described in the above section, is responsible to match players to play the game. First, a player is selected from the pool at random. Then, a predefined matching function is used to calculate the most suitable opponent. Both of these players are removed from the player pool and play a game against one another. After this game is completed, the players' Elo ratings are updated, and they are returned to the player pool. This is run for as many times as there are games in a simulation. The pseudocode for this can be found below.

```
SELECT and REMOVE player P from player pool
FOR each remaining player in the pool, $P_i$
    COMPUTE ladder matching function for ($P$, $P_i$)
SELECT P*, the player with smallest matching function
REMOVE P* from the player pool
RUN GAME between P and P*
UPDATE P and P*'s Elo rankings with game results
INSERT P and P* back into the player pool
```

Figure 1: Ladder Pseudocode

Engine Process

The "Engine", which is responsible for running the rules of a game, follows the following process to run each game. The internal game state of the Engine and the Players are initialized. Then, players make their moves simultaneously, and the turn is run with those moves. The Engine's internal game state, as well as the players' game states are then updated. This routine is run until stopping conditions for the game are met. Once the stopping conditions are met, the engine returns the results of the game.

```
INITIALIZE internal game state
REPEAT
    Players make their moves
    RUN the turn with the chosen moves
    UPDATE internal game state
    INFORM players of the updated game state
UNTIL stopping condition is met
RETURN results of game
```

Figure 2: Engine Pseudocode

Player Move-Making Process

For a game of RPS, there are two types of players: Random and CounterRPS. Random players draw a random number and then play whichever of Rock, Paper, or Scissors (encoded 0, 1, and 2 respectively) that corresponds to that number. CounterRPS players will, on the first turn, play one of Rock, Paper, or Scissors with uniform probability. Then, on every other turn, they will play the move that would beat their opponent's last move. That is, if the opponent plays Paper (1), the CounterRPS player will play Scissors (2). The pseudocode for this method can be found below.

```
IF first turn of game
    PLAY one of $\{0, 1, 2\}$ with probability $1/3$
ELSE
    PLAY $(\textit{opponent's last move} + 1) \bmod 3$
ENDIF
```

Figure 3: CounterRPS move logic

In a game of Pokémon, players can either switch to another Pokémon in their party or attack the opponent. Similar to RPS, there are two types of players: Random and BasicPlanning. A Random player will choose to switch or attack, if possible, with probability 0.5. Then, the choice of which Pokémon to switch to or which move to make is chosen by a uniform distribution. A BasicPlanning player is a bit more nuanced in how they play. This player first generates all possible moves that it, and its opponent could make. Next, it calculates the move that maximizes its expected 'battle position' function based on its opponent's possible moves. This battle position is a quantification of the current state of the game, where better situations correspond to higher numbers than worse situations. The pseudocode can be found below.

```
CALCULATE all possible player moves
CALCULATE all possible opponent moves
FOR EACH player move
    FOR EACH opponent move
        CALCULATE game state
    ENDFOR
    AVERAGE game states calculated at each opponent move
ENDFOR
RETURN optimal move
```

Figure 4: BasicPlanning move selection method

Design Concepts

Basic Principles

This design assumes that players are rational and will choose a more beneficial move over a move that will hinder them. Taking this further, it assumes a player will always think through every possible situation each turn. Finally, these players are operating under incomplete information.

Objectives

The BasicPlanning Pokémon agents are using the objective below defined below.

$$Battle\ Position = \frac{\%\text{ Remaining Hit Points}}{\%\text{ Opponent}'\text{s Remaining Hit Points}}$$

This function is meant to balance advancing towards winning, which happens when the Opponent’s remaining hit points reach zero, and not playing recklessly, which can be quantified by the remaining hit points a player has. This is a very simple form of quantifying the game’s battle state, but is sufficient for now. A more nuanced version of this formula would perhaps weight certain Pokémon’s health as more valuable if they are more threatening to the opponent’s team, or conversely more threatening to the user’s team, which would result in the Position changing more drastically based on which Pokémon in specific lose more health.

Observation

Logging is performed at three levels in this simulation—at the move, game, and overall metagame level. Each turn of a Pokémon battle is logged, recording the current game state, and the choice that was made. However, this output is suppressed since as it stands there is no effective way to analyze this information. The outcome of each game is recorded, with the players’ types as well as their Elo rankings. Finally, the average Elo ranking across agent types are recorded during the simulation, to give an idea for the overall metagame development.

Emergence

For a CounterRPS agent, they play Rock, Paper, or Scissors based on their opponent's last move. Therefore, the probability that they play Rock against a specific opponent, can be calculated using the formula below.

$$\Pr(Play\ Rock) = \sum_{X \in \{Rock, Paper, Scissors\}} \Pr(\text{Play Rock} \mid Opponent's\ plays\ X) * \Pr(Opponent's\ plays\ X)$$

However, since it will only play Rock after the opponent plays Scissors (or play Paper after the opponent plays Rock), the strategy is defined using the calculations below:

$$\Pr(Play\ Rock) = 1 * \Pr(Opponent's\ plays\ Scissors) + 0 * \Pr(Opponent's\ plays\ Paper) + 0 * \Pr(Opponent's\ plays\ Rock)$$
$$\Pr(Play\ Rock) = \Pr(Opponent's\ plays\ Scissors)$$

The probability that the agent plays Paper is therefore the probability the opponent plays Rock, and the same logic can be applied to calculate the odds it plays scissors. Therefore, against an opponent who plays exclusively Rock, for example, it will play exclusively paper. This begs the question, what strategy does a CounterRPS player adopt against another CounterRPS agent?

Recall that on the first turn the CounterRPS agent plays one of Rock, Paper, Scissors with uniform probability, so on the second turn the probability that it would play rock is

$$\Pr(Play\ Rock) = \Pr(Opponent's\ plays\ Scissors) = \frac{1}{3}$$

The same results can be calculated for the probability that player plays Paper or Scissors; the player retains the uniform probability of playing any of the moves. This suggests that the best way of playing against an opponent who plays randomly is to play completely randomly oneself.

Input Data

This project uses two main sets of external data: (1) Pokémon and Move data and (2) Pokémon usage statistics data. The Pokémon and Move data come from the PS github repository, and are used to calculate the statistic values, damage, and effectiveness. The usage statistic data comes from the online Pokémon community Smogon, which details what moves and stat investment the users were using on a month-by-month basis. These data are used for agent prediction under uncertainty, such as guessing at moves that have not yet been revealed.

Sub-models

Damage Stats

"Damage Stats" are an approximation used to estimate how much damage a Pokémon will do without needing to know the opponent's exact statistic values. This relies on the fact that most users either use full investment (252 EVs) or none (0 EVs), and that the damage done in these two cases is different enough that exact calculations are not necessary to determine their difference. These calculations are used by the agents to approximate the opponent's set, based on the damage taken or dealt at a specific turn.

Updating Internal Game State

When a Pokémon Agent gets the information from a turn, they figure out which possible combinations of attack and defense investment from the opponent could result in this move's outcome. This information is then compared to what the agent already knows, and their inference on the opponent's set is updated. The pseudocode can be found below:

```
CALCULATE all possible combinations of opponent's investment
FOR EACH calculated investment
    ESTIMATE damage range given the investment
    IF actual damage is in damage range
        STORE this investment
    ENDIF
ENDFOR
UNION new possible investments and possible investments
```

Figure 5: Inference on opponent's set

Results

RPS Ladder Simulations

In a competitive setup of a game such as Rock-Paper-Scissors, one would expect that as one strategy dominates, all that a player needs to do is play in a way that counters this dominant strategy and one would be able to climb the rankings easily. In many competitive games, it becomes appealing to play in a way that beats what is common in the metagame—to play anti-metagame strategies. This hypothesis was tested against two different player matching schemes: a Random Ladder which randomly pairs players, and a Weighted Ladder which pairs players based on Elo rankings. Simulations were run for 50,000 iterations, with 12 Rock, Paper, and Scissors agents. Games in the simulation were "Best of 5".

In the unweighted ladder case, a heatmap representation of the number of games the agents played (Figure 6) against one another shows that all the different types of players are playing each other roughly the same amount of times, with the number of games ranging between roughly 10,500 and 11,400. This behavior is expected, since there is no bias in the ladder's pairing of players.

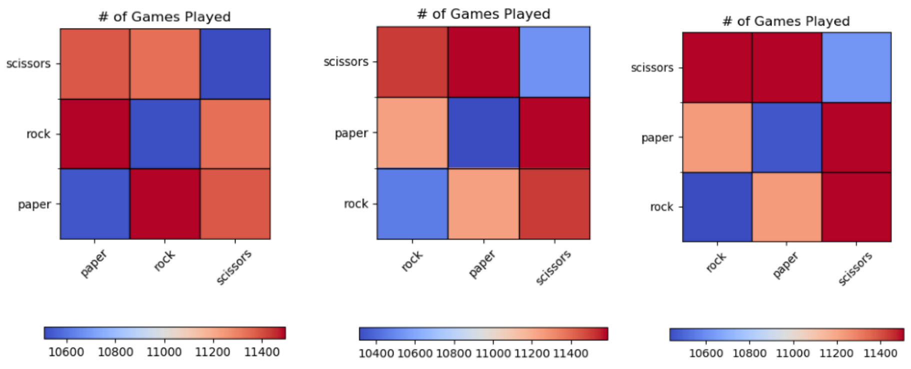

Figure 6 Heatmaps of number of games played between agent types

As expected, across these simulations the ratings of the players oscillate because as it the Rock players' ratings increase, it becomes more valuable (from an Elo gaining perspective) to play scissors.

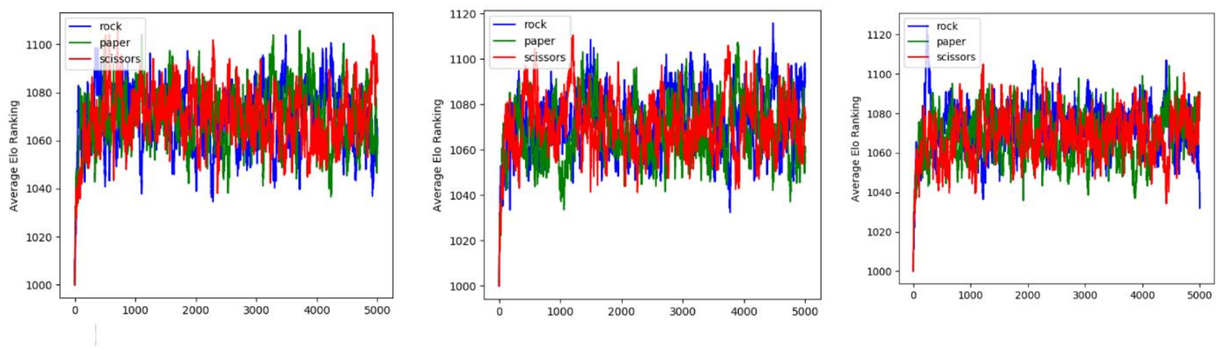

Figure 7: Elo rankings over time for Rock, Paper, Scissors players

However, the results change significantly when the matching scheme changes. Analyzing the heatmaps of the number of games played reveals that the agents predominantly play players of similar strategies to themselves, which means that even if it becomes more beneficial to play scissors, it doesn't matter because the scissors players are only playing against each other.

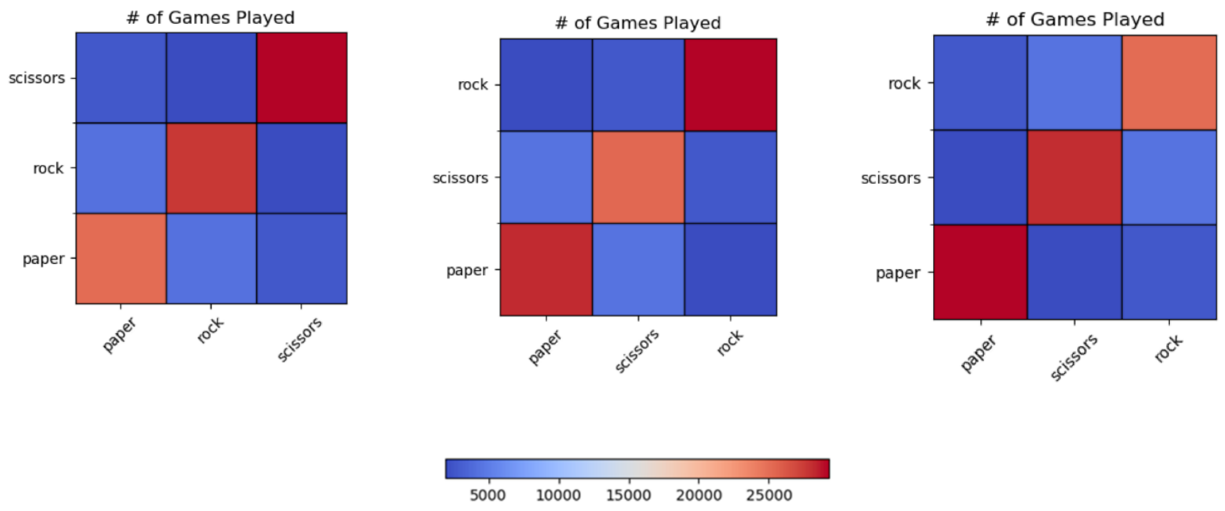

Figure 8: Heatmap for number of games played in Weighted Ladder Scheme

The explanation for this behavior can be found by analyzing the Elo rankings of the players over time, where strangely enough the mixing behavior is no longer observed. Instead, the agent rankings section off, and the Elo rankings stabilize.

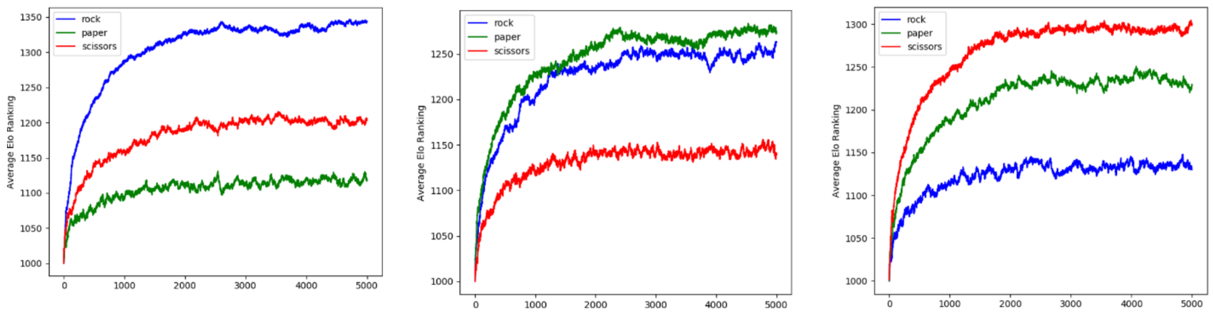

Figure 9: Elo rankings over time in Weighted Ladder scheme

These results indicate that the scheme used to match players fundamentally changes how the metagame develops.

RPS Strategy Simulations

In this simulation, a pool of Rock, Paper, and Scissor agents are pit against a few CounterRPS players. The simulation was run using a Weighted Ladder scheme, and the intention was to demonstrate the CounterRPS agent's theoretical strategy. The simulations were run for 50,000 iterations, with 12 Rock, Paper, and Scissors agents, as well as 2 CounterRPS agents.

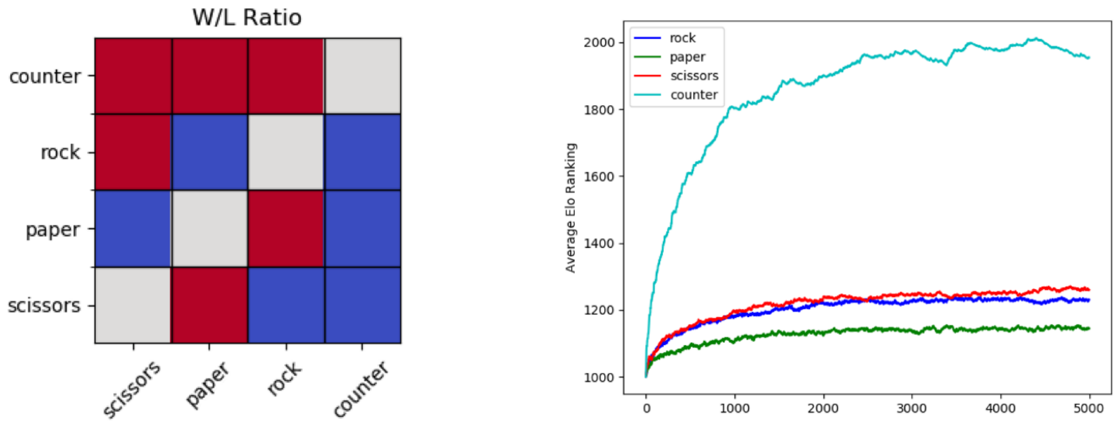

Figure 10: Simulation results with CounterRPS agents

This heatmap represents the matchup matrix, where cells are the proportion of winning, with red corresponding to 1.0, and blue responding to 0. Immediately we see that the CounterRPS agents dominate any Rock, Paper, or Scissors players they play against, and preform 50% against their own kind. These results show up in the Elo rankings as well, where the CounterRPS agents Elo rating is significantly larger than any other player in the simulation.

Pokémon Simulations

This simulation considers three Pokémon; Floatzel, Ivysaur, and Spinda (Figure 11). Generally speaking, Floatzel beats both Spinda and Ivysaur, and Ivysaur and Spinda are evenly matched. However, Ivysaur has a much better matchup against Floatzel than Spinda does.

Figure 11: From left to right, Floatzel, Ivysaur, and Spinda

This setup is used to explore the ability for a player to “outplay their matchup”; can a “better” player using an inferior Pokémon beat a “worse” player using something superior? In order to test this, these Pokémon were assigned to both Random and BasicPlanning Pokémon players—one would expect the BasicPlanning player using Floatzel to do best and the Random player using Spinda to preform the worst, but what happens in the intermediate stages? Is the difference between Floatzel and Spinda too much for “skill” to matter in the outcome?

Simulations were run for 50,000 games, with 10 players of each type (60 total players). Since the question of interest is how the players fare against one another, it is important that they all play each other enough times, and therefore a Random Ladder was chosen.

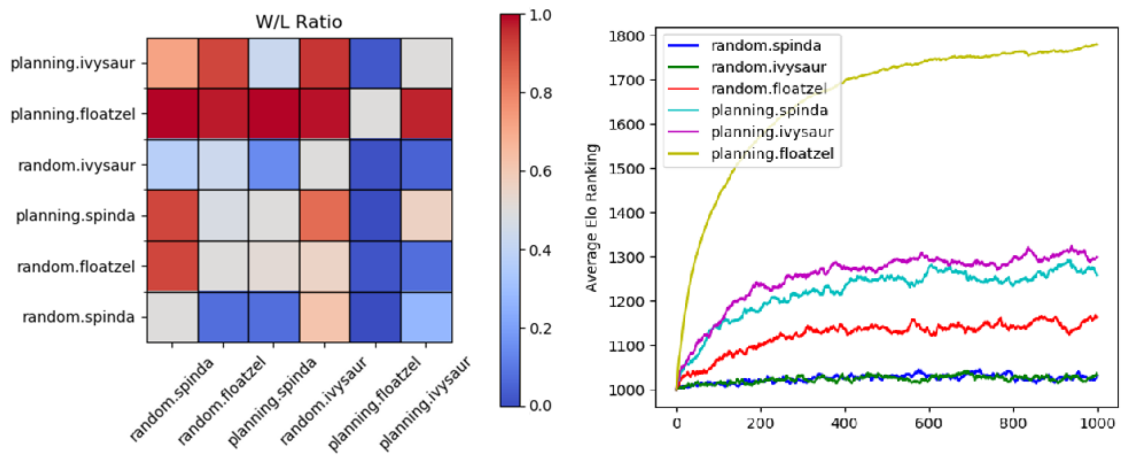

Figure 12: Match outcome heatmap and Elo graph for Pokemon Simulation

Looking first to the results of the match outcome heatmap, the first row that stands out is that Planning Floatzel beat every other player in the simulation. Somewhat surprising is that

Random Spinda seemed to outperform Random Ivysaur, and seemed to have an edge in that matchup. Planning Spinda and Ivysaur both performed well against their Random counterparts. It is in their matchup against the Random Floatzels that we see Plannnig Ivysaur's edge; since it has the means to reliably beat Planning Floatzel, it tends to do better.

In this simulation, the Elo rankings are indicative of overall "strength", since they represent the overall matchup against other players in the metagame. Unsurprisingly, Planning Floatzel dominates, followed by the Planning Ivysaur and Spinda players. Random Floatzel managed to differentiate itself from the other random players but is not good enough to get past the planning agents. Therefore, we see that the strategic players are overall more fit than the random players, even if those players are using superior Pokémon.

Lastly, this simulation run against a Weighted Ladder, while it does not change the overall matchup structure, does develop an oscillation between the Planning Ivysaur and Spinda agents, much like that in the Random Ladder RPS Simulation, as can be seen in Figure 13. While the overall picture is the same, it is an interesting result and supports the finding that the matching scheme can alter the overall metagame.

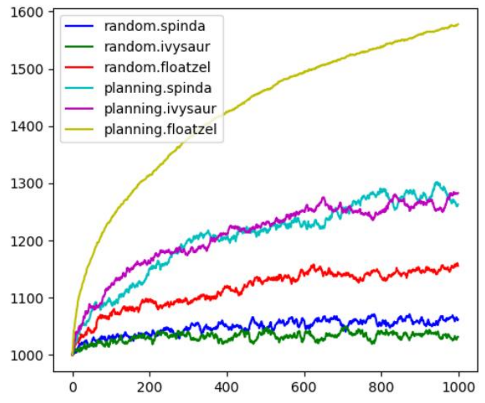

Figure 13: Pokemon Simulation with a Weighted Ladder

Technical Details/Running the Project

This report is written using version 1.2.1 of the code. This release can be found at the following link: *https://github.com/aturfah/cmplxsys530-final/releases*

The code in this project was written for Python 3. The necessary packages are contained in the *requirements.txt* file—in order to install them run 'pip install -r requirements.txt'.

Simulations are managed by the *run_simulation.py* file, the details of which can be accessed by typing 'python run_simulation.py -help'. Premade simulations are in the *sample_simulations* folder. Once a simulation is run, it will generate some number of log files in the *logs* directory—they will be in csv format.

As a general rule, log files with 'Players' tags in them are the match outcomes, which can be used to generate the matchup matrices. Those with the 'Type' tags are the overall metagame Elo measurements, and can be used to generate the Elo graphs.

Finally, should someone want to play against a Pokémon Agent, the *run_interface.ps1* script in the *scripts* directory will start up a local server where agents can be played against directly. This server can be accessed at the address “localhost:5000” from a web browser. This script is written for Windows machines, I’m not sure if it will run properly on a Mac.

Future Directions

In future work I would like to improve the representation of the Pokémon’s game engine—status effects, boosting, entry hazards, among others. This would allow for a more accurate representation of the actual Pokémon metagame. In addition, developing different ‘battle position’ functions would be helpful in order to get a more precise representation of the different Ladder segments.

References

Smogon User "X-Act", S. U. (2009, October 18). *Damage Stats - An Easier Way to Calculate Damage*. Retrieved from Smogon Unviersity: https://www.smogon.com/smog/issue4/damage_stats

Anderson Tavares, H. A. (2016). Rock, Paper, StarCraft: Strategy Selection in Real-Time Strategy Games. *The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* , (pp. 93-99). Phoenix.

Glickman, M. E. (1995, April 22). Chess Rating Systems. *American Chess Journal*, 59-102. Retrieved from Mark Glickman's World: http://www.glicko.net/research/acjpaper.pdf

Peter Cowling, M. B. (2013). *Search in Real-Time Video Games.* Leibniz-Zentrum: Dagstuhl Publishing.