Exploring AI and Statistical Physics with Pokémon

Hyunsuh Gu, Delta Blendea, Jessica Liu, Avi Iyer, Atreyi Saha,
and Mentor Sofia Helpert

Abstract

While artificial intelligence (AI) can be incredibly powerful when outcomes are consistent, they will often fail to perform as consistently when randomness and luck play a large factor in their environment. Examining the Brock fight in Pokémon Red, we compared a model that chose moves randomly with a reinforcement learning (RL) model to map the battle landscape and better understand how such models adapt to random situations. The RL agent was ultimately able to win about 95% of games, compared to the roughly 45% win rate of the random agent. We found that on average, both models win in 32 turns, however under certain conditions the RL model can win in as few as 28 turns. Like the random simulation, the RL model found that early growl usage was imperative to winning the battle, however unlike the random simulation, the RL model was able to fine tune the number of growls it used in order to win the battles faster.

I. Introduction

August 9, 2024

THE Pokémon gaming community presents to academia a fascinating study in the realm of gaming and AI, particularly the online gaming community of streamers, YouTubers, and their viewers. Unlike the traditional role of AI in games such as Go or chess, where strategies are predominantly deterministic, what differentiates Pokémon battles is its characterization by a significant degree of unpredictability.

This investigation seeks to explore the performance of AI algorithms in Pokémon battles that contrast to other traditional games on the basis of randomness. Through battle simulations and analyzing game paths, we aim to understand the element of randomness that shapes the landscape of the battles by applying principles of statistical physics. Additionally, we also aim to assess the extent to which "luck" influences battle outcomes and whether there is a certain threshold of luck necessary to meet in order to win a Pokémon battle/achieve success. We hope to contribute to uncover potential strategies to enhance the gaming experience by utilizing statistical physics and artificial intelligence.

A. AI/reinforcement learning in other games

AI has been used in many other games, such as the Deep Blue AI for chess as early as 1996. However, Pokémon differs substantially from strategy-heavy games such as chess because large portions of the game are entirely random. As such, it's difficult to compare largely deterministic algorithms and data-driven AI models since their methods are very different. Nevertheless, examining data and visualizations from other games (e.g. chess, go) proved helpful as many patterns can be observed which AIs can be built on. For instance, the probability of a chess piece moving on a given move can be likened to the probability of different moves being used in Pokémon.

Reinforcement learning models have previously been applied to Pokémon Red [1], [2]. These models use the screen data to produce game inputs to be sent. While this approach does not limit the potential outcomes of the model, overloaded outputs (where the effect of a button depends on context) make global generalization difficult. There is also an instance of reinforcement learning being applied to a Pokémon battle simulator, where a controlled environment allowed the authors to isolate specific objectives [3]. Their environment provided a higher-level interface for the model, where the output options were the 4 attacking moves and a switch. We will attempt to explore the space between the two, where battles are played out by an emulator, but the model still interacts with the environment through a high-level interface.

B. Visualization in games and AI

To get a better understanding of how our Pokémon data visualizations could potentially look like, we first took a look at existing visualizations for Pokémon and other similar games, like chess and Go [4], [5], [6]. Statistical graphics breaking down gameplay decisions, such as heat maps of the most played tiles and pieces or plots depicting the probability of a piece being moved help us better understand general trends regarding the game. We can adapt these ideas to be more relevant to our scenario, such as depicting the probability of a move being used instead of a piece being moved.

II. Statistical Physics in Pokémon and AI

The integration of statistical physics in the realm of Pokémon gaming presents a frontier for enhancing gameplay strategy optimization. At the core of this interdisciplinary approach lies the recognition that Pokémon battles can be viewed as complex systems with many interacting components. Each turn in a Pokémon battle can be described as a microstate, a detailed configuration encompassing the exact health points, status conditions, move sets, and positions of each Pokémon, as well as field conditions and turn counts. The number of possible microstates in a typical battle is astronomically large, making direct enumeration and analysis computationally intractable. This is where the concept of macrostates becomes imperative. Macrostates in Pokémon represent higher level descriptions that aggregate many microstates, such as the number of Pokémon remaining on each team, the general health status of the teams, or the overall situation. Working with macrostates, we can significantly reduce the complexity of the game state space while retaining meaningful strategic information.

Entropy, a fundamental concept in both statistical physics and information theory, offers a powerful tool for quantifying the uncertainty and information content in Pokémon battles. We can define a ”battle entropy” as a measure of the unpredictability or ”chaos” in a given game state. High-entropy situations might represent volatile battle conditions where the outcome is highly uncertain, while low-entropy states could indicate a clear advantage for one player. By tracking changes in entropy over the course of a battle, we hope to identify critical moments where small actions can have a significant impact on the game's outcome.

To analyze microstates and macrostates in Pokémon battles, we define:

- Microstate: A specific sequence of events at a microscopic level, including individual moves, critical hits, misses, and health points.
- Macrostate: The overall outcome of a battle, such as victory or defeat, derived from various microstates.

III. Methods

We used a PyBoy emulator and Python script to record data about the Bulbasaur/Brock battle, including moves used, HP, and status effects. In our first baseline test, we had our agent select random moves, beginning with leech seed in order to slightly increase the chances of winning. Afterwards, we developed a reinforcement learning model and had it train over about 100,000 games. We then compared the data of these two simulations to determine how much more effective our model was at defeating Brock than the baseline model.

A. Brock/Bulbasaur battle details

Brock is the leader of the Pewter Gym, marking the first milestone progression in Pokémon Red. Brock brings Geodude and Onix, both of which must be defeated in order to win the fight. For simplicity, we entered the fight with only the starter Bulbasaur at level 7, the minimum level required to challenge Brock. This limits the output space to Bulbasaur's three available attacks: Tackle, a standard damage-dealing move; Growl, which decreases the opponent's ATK (attack) stat; and Leech Seed, which plants a seed on the opposing Pokémon until it faints or is swapped out. Seeded Pokémon have their health drained and heals the original attacker every turn.

Human players have found an almost perfectly consistent winning strategy for this matchup [7]. Because seeded damage is only dependent on the opponent's max HP and also provides consistent healing, it is possible to wait out the battle by first using leech seed followed by 6 growls (which leads to the maximum reduction in ATK) to reduce incoming damage. However, while consistent, this strategy is very slow, and we believe that using growl slightly less and using tackle slightly more can lead to still consistent victories that also take less turns, saving time for speedrunners and challenge runners.

B. Simulation Details

Pokémon Red was originally released for the GameBoy, a handheld console by Nintendo. In order to play Pokémon Red, we needed an emulator, or a piece of software that can simulate older hardware to run applications or play games on, that we could interface with through Python code.

For our game environment, we used PyBoy [8]. We decided to use PyBoy because it has tools for accessing and modifying the emulated memory directly, in addition to prioritizing performance for fast training and simulation running. PyBoy also allows us to further increase the simulation speed by running the emulators headlessly, reducing the need to process unnecessary graphical information, as well as frame skipping, reducing the amount of unnecessary work the program does. With these optimizations, we were able to fully record 400,000 Brock/Bulbasaur battles in a little over 12 hours on a desktop PC, resulting in over 600 MB of data detailing exactly how each of those fights went. We recorded moves used, HP after every move, whether the move was a critical hit or a miss, and status effect information, and information about the battle itself, such as the length and whether or not our Bulbasaur won.

The data was gathered by directly accessing the emulated memory. Thanks to the work of ROM hackers and other enthusiasts, RAM addresses were available for most in-game values and flags [9]. We verified these values with our own testing. With these values, we could at any point access values such as Pokémon HP, status effects such as poison, burning, or leech seed, and details about what move was just used by the enemy Pokémon.

C. Reinforcement learning details

Reinforcement learning is a type of machine learning which is used for a variety of tasks, but is particularly powerful in applications such as playing games. In reinforcement learning, an "agent" receives information about the "state" of the environment, and uses that information to predict which action would maximize its "reward'. For example, if a reinforcement learning agent was learning to play chess, the state might be the positions of pieces on the board and whose turn it is, and since capturing the enemy queen would lead to a higher chance of victory, that action would result in the highest reward.

To implement our own reinforcement learning model to play Pokémon Red, we used a Python script and the Gymnasium library [**?**]. Firstly, we created our own custom Gymnasium environment. This environment defines concepts that Gymnasium uses, such as the observation space, or the set of all possible observations the agent can expect to encounter, and the action space, the set of all possible actions the agent can use. We used 5 discrete spaces to create the action space, Bulbasaur's HP, Geodude's HP, Onix's HP, whether the enemy Pokémon is seeded or not, and the attack stat stage of the enemy Pokémon (or, how many growls were effectively used). The action space is a single discrete space from 1 to 3, representing the 3 different moves Bulbasaur can use. We then created a translation layer for the PyBoy emulator that would turn basic commands, such as "use attack #2", into a series of button presses ("a", "down", "a"). This would allow us to directly apply an action that is chosen by the agent into the PyBoy emulator.

In our case, the reinforcement model used an observation containing: Bulbasaur's HP, Geodude's HP, Onix's HP, the attack stat stage of the enemy Pokémon (effectively how many successful growls Bulbasaur has used), and whether or not the enemy Pokémon has the Leech Seed debuff applied. While training, the agent will use its observation to check if the state it is currently in is the same as a state it has encountered before. If so, it will choose whichever move has previously led to the highest win rate from that position. If it hasn't, it will pick a move at random. In addition, we implemented an epsilon value of 0.3, which means that even if the agent has encountered a state before and has an idea of the best move, there is a 3 in 10 chance that it will pick a move at random instead. This is to limit the chance that the agent gets "locked in" a certain play style, ignoring potentially better paths. Our reward function was calculated as: If the battle is won, the reward is $1 - n_{turns}/100$, where n_{turns} is the number of turns it took to win the battle. If the battle is lost, the reward is 0. This ensures that the computer will first optimize for wins, then further optimize for the fastest wins possible.

D. Statistical Physics

Here we define two types of microstates that we will be considering in our work: "battle paths" and "HP pairs". A battle path is an ordered list of moves that represent both the player and opponent turns. Bulbasaur's moves include leech seed, tackle, and growl, while Geodude has tackle and defense curl and Onix has tackle, screech, and bide. Since tackle is the only move that can land a critical hit we also consider "crit tackle" and finally "miss", which represents when a move does not hit, whether because of the player's accuracy or the opponent's evasiveness. The other type of microstate we consider, HP pairs, is an ordered pair representing the player and opponent HP on any given turn.

In our analysis of battle paths, we calculate the entropy of battles of length l, S_l, using the Shannon entropy formula [**?**]:

$$S_l = -\sum_j p_j \cdot \log(p_j) \tag{1}$$

where p_j is the probability of the jth battle path occuring, which is calculated as the number of times battle path j occurs divided by the total number of battles of length l.

IV. Results

A. Random Simulation Results

Figure 1 shows the distribution of battle lengths in our random simulation as well as the average percent move usage for battles of each length. It can be seen that the minimum amount of turns it takes for the random simulation to win is 29 turns and the average is around 32 turns. This indicates that there is a certain amount of setup necessary to win, with wins around battle length 29 hypothesized to be more luck dependent. There is also a lot of variability in the lower length battles, specifically battles of length 6, which is where we attempt to use some of our statistical physics tools to gain a deeper understanding.

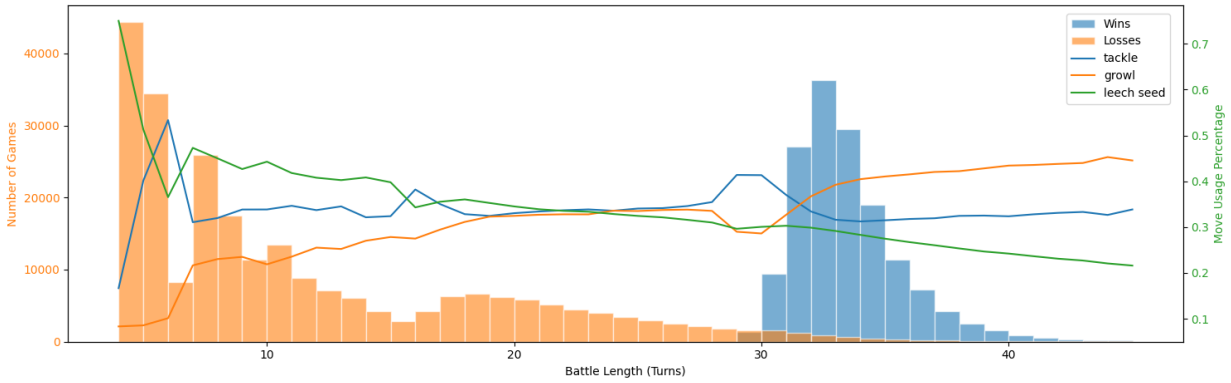


Fig. 1: Distribution of battle lengths and move usage percentage per battle length of the random simulation.

Here we will calculate the total number of possible battle paths given the length of the battle, l. For each turn, Bulbasaur has 5 possible moves: leech seed, growl, tackle, crit tackle, and miss; Geodude has 4 possible moves: defense curl, tackle, crit tackle, and miss; and lastly Onix also has 5 possible moves: bide, screech, tackle, crit tackle, and miss. Note that although technically leech seed, screech, and tackle can all lead to a miss, we group them into a single miss move which indicates that there was no change in HP, status effects, etc.

Thus in a single turn against Geodude there are $5 \times 4 = 20$ possible paths and in a single turn against Onix there are $5 \times 5 = 25$ possible paths. Now we define the number of turns against Geodude, l_G, and the number of turns against Onix, l_O, such that

$$l = l_G + l_O \tag{2}$$

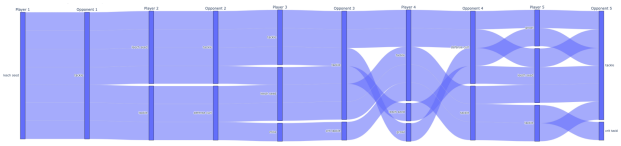
Then we can calculate the total number of possible battle paths of length l, N_l as

$$N_l = (20)^{l_G} \times (25)^{l_O} \tag{3}$$

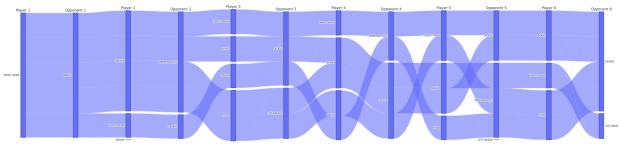
This formula is true for our RL agent since its first turn is unconstrained, however since the first turn of the random simulation is fixed (Bulbasaur always chooses leech seed, Geodude always chooses tackle) the total number of possible battle paths in the simulation is given by

$$N_{l,sim} = (20)^{l_G - 1} \times (25)^{l_O} \tag{4}$$

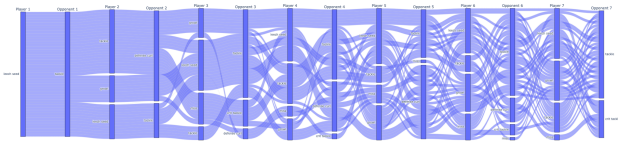
For illustrative purposes, let us consider a battle of length $l = 6$, where assuming we never beat Geodude in less than 6 turns, $l_G = 6$ and $l_O = 0$. Using our formula from above, we find that a 6 turn battle has $N_{l,sim} = 3.2 \times 10^6$ possible battle paths.



(a) $l = 5$



(b) $l = 6$



(c) $l = 7$

Fig. 2: All possible battle pathways of battles length 5, 6, and 7 turns for the random simulation.

It's important to note that while there may be 3.2 million possible 6 turn battle paths, due to HP constraints only a handful of them actually end the battle in 6 turns, while the rest go on to become longer battles. Figure 2 shows battle path schematics for observed battles of length 5, 6, and 7 turns. Interestingly, there are fewer battle paths resulting in a 6 turn battle than there are for a 5 turn battle, which may explain the significant drop in occurrence from 5 to 6 turn battles observed in Figure 1. The typical expected behavior is for the number of battle paths to increase as the number of turns increases, as seen in Figure 2c, and indeed as we continued to increase the number of turns shown in our schematic, the more complex and unreadable it became.

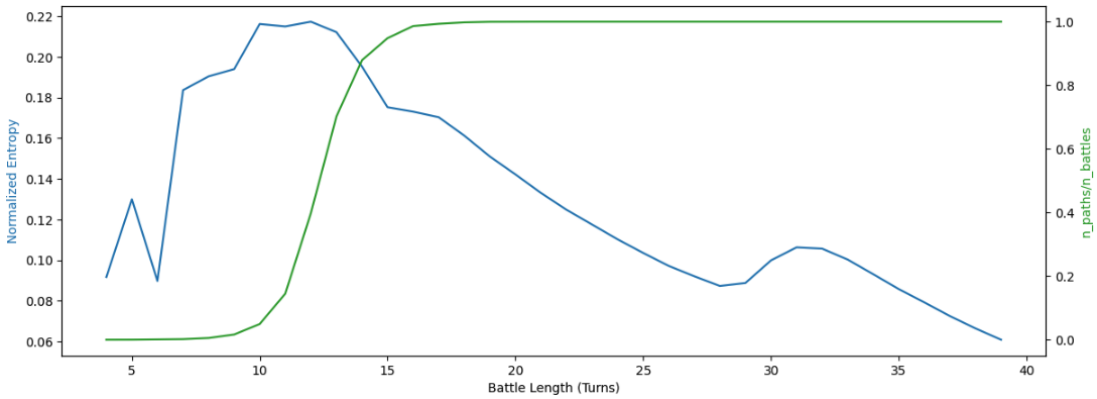


Fig. 3: Observed normalized entropy and the ratio of observed number of battle paths to total number of battles per battle length

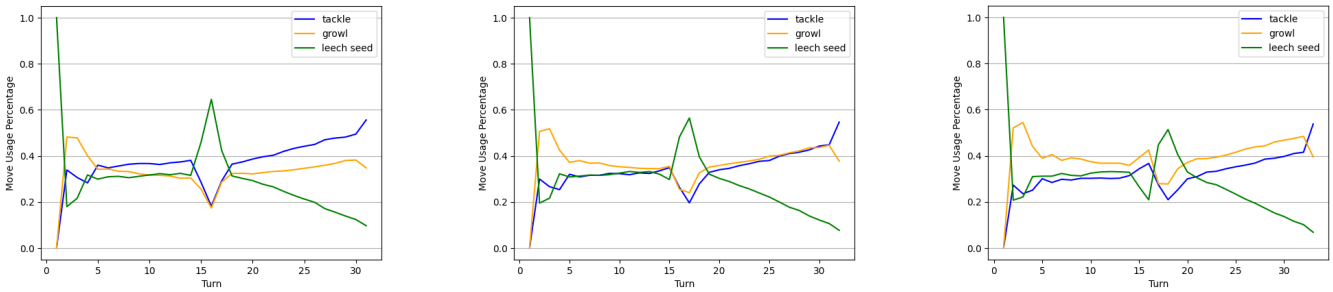
Figure 3 shows the normalized entropy that we observed in all battles of length l, as well as the ratio $\frac{n_{paths}}{n_{battles}}$, where n_{paths} is the total number of unique battle paths we observed of length l while $n_{battles}$ is the total number of battles of length l that we recorded. The normalized entropy was calculated as

$$S = \frac{S_l}{-\log(1/N_{l,sim})} \tag{5}$$

where the denominator represents the maximum entropy, which can be found by assuming a uniform distribution across all possible microstates. Here we assume that $l_G \leq 15$, since in our results we find that Geodude most often faints after 15 turns.

We see in Figure 3 that entropy does indeed decrease from $l = 5$ to $l = 6$, reinforcing our conclusion that the occurrence of 6 turn battles in Figure 1 is so low because there are fewer battle paths. This does not, however, explain why the occurrence of 4 turn battles is so high, given that the entropy is lowest at $l = 4$. In an attempt to explain this, we found that battles always last only 4 turns when Geodude uses tackle 4 times in a row. Given that the first turn is always tackle, and that in each subsequent turn the chance of Geodude using tackle is $\approx \frac{1}{2}$, the probability of Geodude using tackle 4 times in a row is $\approx (\frac{1}{2})^3 \approx \frac{1}{8}$. Since we ran 400000 battles, we should expect ≈ 50000 of them to end in 4 turns, which is what we observe and explains why there are so many 4 turn battles in our data.

We also see in Figure 3 that the entropy peaks in battles of length 9 to 12, indicating that these turns have the highest uncertainty and must be crucial to the outcome of the Geodude battle. However, it's also from battle lengths 9 to 12 that the ratio of observed paths to total battles starts to rapidly increase, and saturates at about $l = 15$, which means that every battle we observe has a different battle path. This indicates that we are most likely underestimating the entropy of battles of length $l > 15$, and that we need to collect more data in order to make conclusions about longer battles.



(a) $l = 31$ (b) $l = 32$ (c) $l = 33$

Fig. 4: Percentage distribution of move usage per turn for battles of length 31, 32, 33 of the random simulation.

Zooming in on battles of specific lengths, particularly those with majority winning games, allows us to visualize what the general winning strategy for the simulation is. From Figure 4, it can be seen that early growls and leech seeding Onix when it comes out are a necessity for success. In the longer battles, the leech seed peak occurring when Onix comes out (or Geodude faints) is later. This can be explained by the heavier tackle usage earlier on in battles of length 31 vs. length 32 and 33, which causes Geodude to faint earlier. Thus, the general strategy for winning games is to use first set up with leech seed and about 3 turns of growl, finishing off Geodude with tackle before repeating the same with Onix.

B. Reinforcement Learning Results

Just generally looking at comparing the overall shape of the histogram in Figure 5 to the histogram in Figure 1, it is clear that the AI wins a lot more; in fact its win rate is about $\approx 95\%$ whereas only $\approx 45\%$ of the battles in the random simulation are winning battles. The average winning battle length for the AI is about the same as the simulation at around 32 turns. However, unlike the simulation, the AI has a few wins of length 28 and averages length 29 wins more consistently, indicating that the AI has found battle possibilities that the random simulation has not. Unlike in Figure 1 where the move percentages are more evenly distributed (except early on, since turn 1 is fixed to be leech seed), in Figure 5 the move usage percentages for battles around length 30 to 35 show that the AI has found an optimal ratio of moves used in order to win: leech seed being $\approx 20\%$, growl being $\approx 35\%$ and tackle being $\approx 45\%$ of all moves used in the battle.

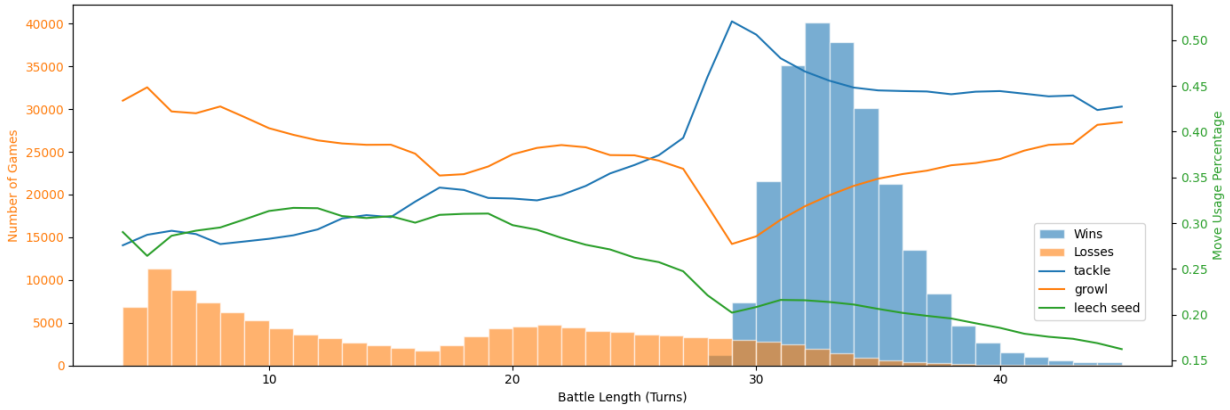
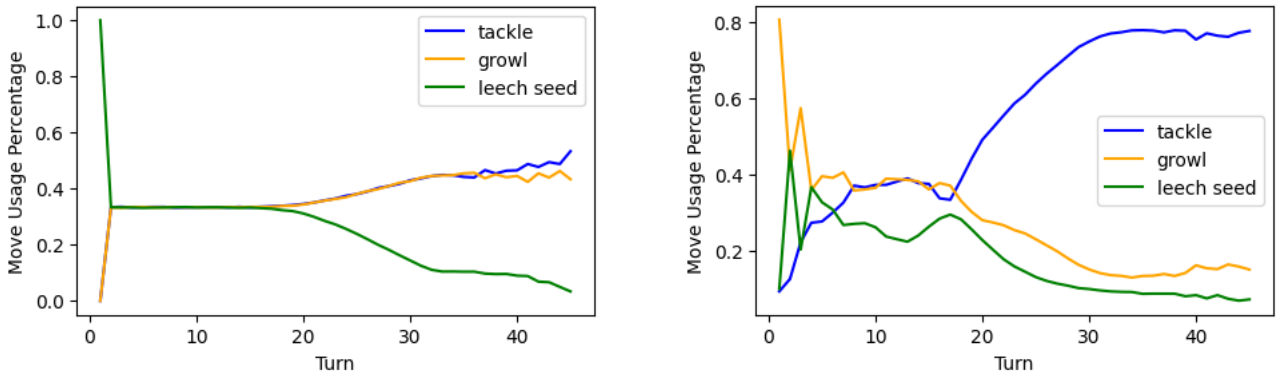


Fig. 5: Distribution of battle lengths and move usage percentage per battle length of the reinforcement model.

In Figure 6(a), each move in the simulation has exactly a 33% usage per turn apart from turn 1, which is hard coded to be leech seed, and later turns when leech seed has started running out of PP (the number of times a move can be used). In contrast, Figure 6(b) shows much more variability in move percentage per turn, showing the RL agent's optimal move sequence. There is heavy usage of growl and leech seed early on for setting up with a switch to almost entirely tackle to finish the battle, matching up with our findings previously from Figure 4.

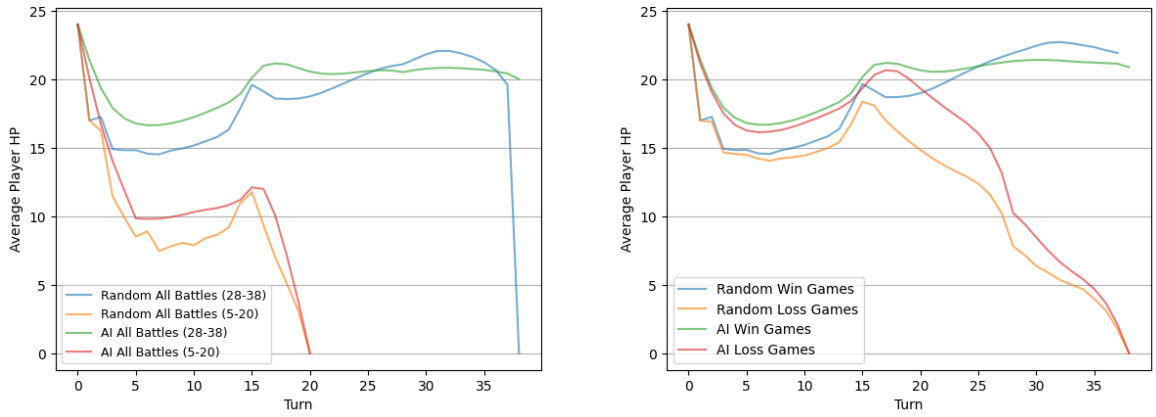


(a) Random Simulation

(b) Reinforcement Learning Model

Fig. 6: Percentage distribution of average move usage per turn for the random simulation and the reinforcement model

In both Figure 7(a) on the left and Figure 7(b) on the left, it can be seen that the AI average player HP follows the same trend as the simulation average player HP. However, the AI model's HP is generally higher and has a smoother curve due to a greater usage of growl lowering the opponent's attack, reinforcing the importance of growl to ensure consistency.

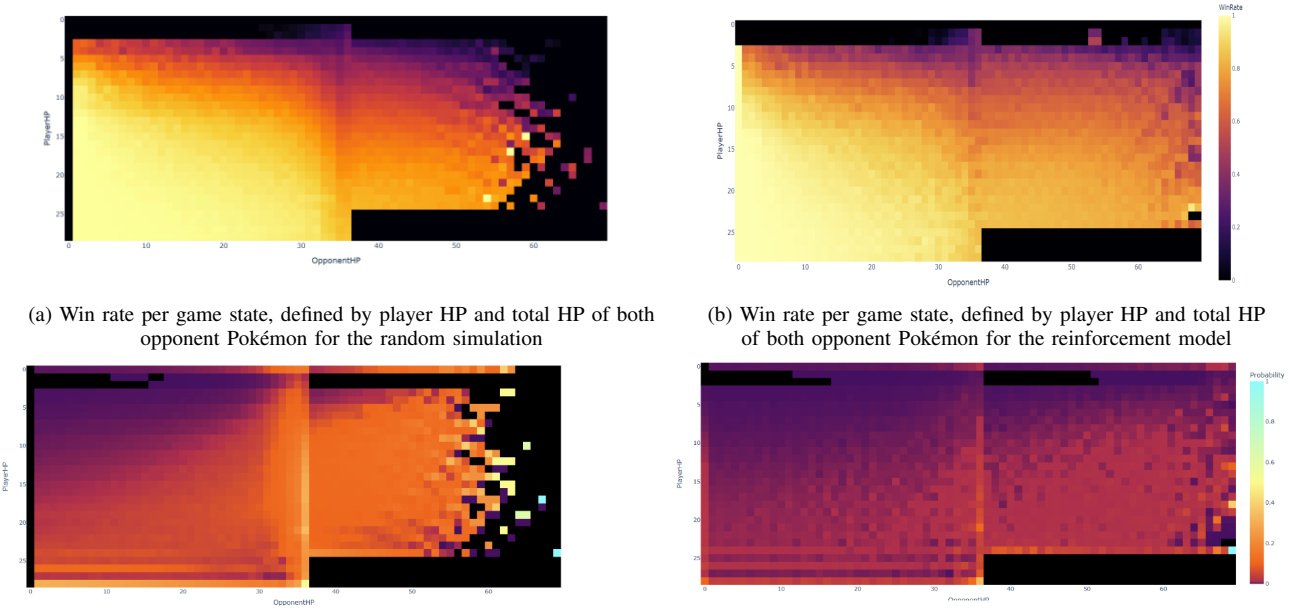


(a) Average player HP per turn for battles length 5-20 and 28-38 of both the random simulation and reinforcement model.

(b) Average player HP per turn for length 28-38 of both the random simulation and reinforcement model, split by winning and losing battles.

Fig. 7: Percentage distribution of average move usage per turn for the random simulation and the reinforcement model

Finally, we consider the other microstate we defined earlier: HP pairs. Figure 8 shows four heat maps displaying the win rate and probability of each HP pair for both the random simulation and the RL model. The win rate is defined as the number of times the HP pair appears in a winning battle divided by the total number of times the HP pair is observed, while the probability is calculated by dividing the number of times the HP pair is observed by the total number of HP pairs observed. The left half of the heat maps show HP pairs from the battle with Onix, who has HP = 36, which is why we see a distinct line here. We also note that Bulbasaur levels up after defeating Geodude, which is why the Player HP ranges from 0 to 28 on the left half and 0 to 24 on the right half.



(a) Win rate per game state, defined by player HP and total HP of both opponent Pokémon for the random simulation

(b) Win rate per game state, defined by player HP and total HP of both opponent Pokémon for the reinforcement model

(c) Probability per game state for the random simulation

(d) Probability per game state for the reinforcement model

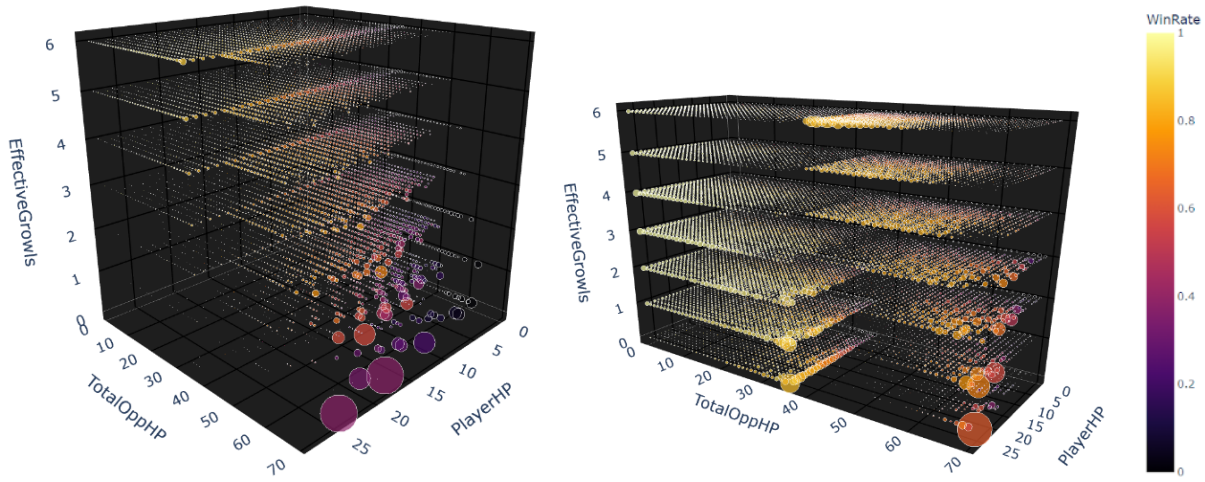
Fig. 8: Heat maps showing the win rates and probabilities for each game state

The win rate at any point in a battle for both the simulation and the AI are about the same as seen in Figures 8(a) and 8(b), however there is less overall purple in Figure 8(b), once again visualizing the AI's higher win consistency. There are also several more light yellow (very high win rate) squares in 8(b), especially where the player HP is very low, indicating that the AI may have developed some strategies to win in such a situation.

In Figures 8(c) and 8(d), we see that although there are 792 HP pairs in the Geodude battle and 1008 HP pairs in the Onix battle, they are not uniformly distributed. The light blue corner in the bottom right of both heat maps represents the starting point, which is why we see high probability here. In general, the RL model remains in regions of higher player HP, which

corroborates our conclusion from Figure 7. We also see that the random simulation has much higher probability in the Geodude fight than the RL model, which can be explained by all of the losses at low battles lengths in Figure 1 compared to Figure 5.

Lastly, we extend the heat maps from Figure 8 by adding "effective growls" as a third dimension, since it has been established to be a significant part of the strategy. In Figure 9(a), as expected, there are generally more and more occurrences of winning games states as the number of effective growls increases, as seen by the larger points. However, in Figure 9(b), the largest points in winning game states, or game states where the opponent HP is zero, actually occurs closer to 3 to 4 effective growls. This is indicative of the AI's reward function, which encourages shorter battles, thus forcing the AI to figure out an optimal number of growls rather than an excessive amount.



(a) Random simulation (b) RL model

Fig. 9: 3D heat maps showing the win rates and probabilities for each game state. The probability of a game state is represented by the size of the point.

V. Conclusions

We found that although both the AI and random simulation on average win after 32 turns, the AI has a wider winning distribution and can win in as little as 28 turns. We concluded that we need to run more battles in our random simulation to better understand the Onix battle, however we found that turns 9-12 seem to be the most critical in the Geodude battle. We found that the winning strategies for both the AI and the random simulation are similar, with early growls and leech seed followed by tackles, however it seems that the AI has somewhat more refined strategies, including limiting its use of growl, that allow it to keep its HP higher throughout the battle.

VI. Further Research

We have a variety of paths to explore in future efforts on this project. Firstly, we would like to implement per-turn rewards rather than per-battle rewards, which, with an appropriate reward function, might speed up the training process and make it even more accurate. In addition, we used exactly one Pokémon with only one set of IV's, which, while easier to train, isn't as impressive as a model capable of adapting to any set of IV's. For example, it would be interesting to see the model using the high defense stat of a particular Bulbasaur to use Growl less or a high attack stat being used to help tackle finish off a Pokémon instead of leech seed. We would also like to improve the model itself by integrating a neural network with Proximal Policy Implementation, which would greatly improve both the speed of training the model and the ability for it to learn the complex nature of Pokémon battles. We would also like to see our model learn different battles with even more complexities, like move effectiveness, having multiple Pokémon to switch between, managing and using items, and more. Finally, we still believe there is more insight to be gained even in our current results, in particular in understanding the critical turns of both the Onix and Geodude battles, how these are governed by random events such as critical hits or misses, and taking a closer look at how our current model navigates these situations.

If you have interest in working on this project, please contact shelpert@stanford.edu for further discussion, or apply to future years of the SHTEM internship!

ACKNOWLEDGMENT

The authors would like to thank the SHTEM Internship Program for the resources and guidance for our project, and Professor Tsachy Weissman for starting this program. In particular, we would like to thank David J. Florez Rodriguez for his endless positivity and support throughout the process. We would also like to extend a massive thank you to Sofia Helpert, our mentor for this project, who helped us along every step of the journey. Without her, none of our research would have been possible. Thank you!

REFERENCES

[1] P. Whidden, "PWhiddy/PokemonRedExperiments: Playing pokemon red with reinforcement learning." [Online]. Available: https://github.com/PWhiddy/PokemonRedExperiments
[2] J. Flaherty, A. Jimenez, and B. Abbasi, "Playing pokemon red with reinforcement learning." [Online]. Available: https://scholarworks.calstate.edu/downloads/vq27zt20r
[3] D. Simões, S. Reis, N. Lau, and L. P. Reis, "Competitive deep reinforcement learning over a pokémon battling simulator," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2020, pp. 40–45.
[4] E. Zhang, "Visualizing chess game length and piece movement," Jun 2021. [Online]. Available: https://www.kaggle.com/code/ironicninja/visualizing-chess-game-length-and-piece-movement
[5] J. K. Fuentes, "Visualizing chess data with ggplot," Oct 2015. [Online]. Available: https://jkunst.com/blog/posts/2015-10-30-visualizing-chess-data-with-ggplot/who-captures-whom
[6] J. Mussalli, "Create and analyze a game graph for pokémon battles," 2023. [Online]. Available: https://community.wolfram.com/groups/-/m/t/2962470
[7] Jrose11, "Is bulbasaur the best kanto starter? - solo pokemon red/blue challenge," 3 2020. [Online]. Available: https://www.youtube.com/watch?v=SObhZKg71ic
[8] Baekalfen, "GitHub - Baekalfen/PyBoy: Game Boy emulator written in Python — github.com," 3 2024. [Online]. Available: https://github.com/Baekalfen/PyBoy
[9] DataCrystal, "Pokémon Red and Blue/RAM map," 6 2024. [Online]. Available: