Percymon: A Pokemon Showdown Artifical Intelligence

SUNet ID: [h2o, vramesh2]
Name: [Harrison Ho, Varun Ramesh]
Repository: github.com/rameshvarun/showdownbot

1 Introduction

Pokemon is a popular role playing game series in which players train a team of six creatures to fight opponents. While the Pokemon games feature a story-line and AI opponents to fight through, an increasingly large online community has focused purely on competitive play. Battles in these communities involve the use of advanced moves and strategies that are not necessary while fighting against the game's rudimentary AI.

To streamline the process of competitive battling, various fan-made simulators have been created to play competitive battles online. The most commonly used simulator is Pokemon Showdown, which regularly has over 12,000 users logged in and playing games [1]. Showdown offers a variety of formats. The "Random Battle" format assigns teams of six random Pokemon to each player - distributions of stats, abilities, types, and levels are randomized, but designed to be fair. Focusing on the Random Battle format allows us to ignore the team-building aspect of competitive Pokemon and to only deal with strategies involved with battling.

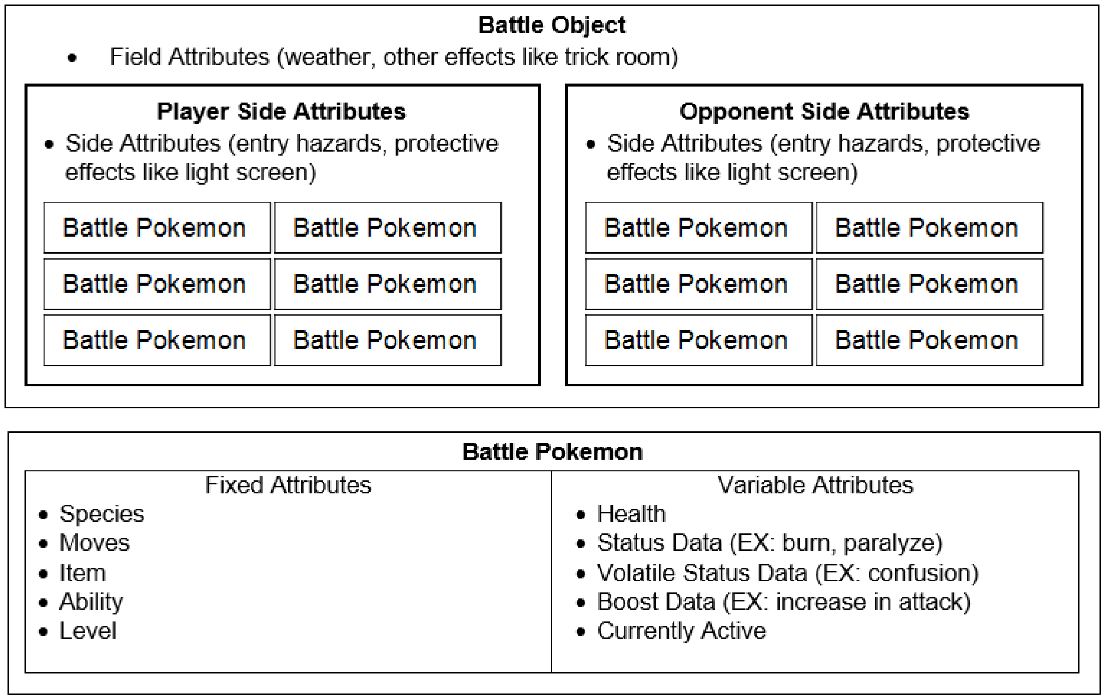
For this project, we developed an AI bot that logs into Showdown and plays games on the official ladder. We implemented a Minimax search of depth 2, and tested the results against a greedy algorithm and a human player.

2 Related Work

Technical Machine is a Pokemon AI written by David Stone and developed for the fourth generation of Pokemon [2] (Pokemon Showdown currently supports the sixth generation.) Technical Machine uses an Expectiminimax search of depth 3. To evaluate states, the evaluation function looks at a variety of factors of the battle and of each Pokemon, multiplying by a manually tweaked weight and summing the weighted factors. To improve performance, Technical Machine uses a transposition table to prevent the re-evaluation of identical states. Technical Machine is different in that it attempts to choose the best six Pokemon to make up a team, while Percymon ignores the team-building step and focuses solely on battling. In addition, Percymon deals with game mechanics unique to Generation 6, which increases the possible state space.

3 Task Definition / Game Model

We model a Pokemon battle as a two-player, zero-sum game. A state consists of all of the information required to simulate moves. The state is encoded inside a Battle object, which has several components:


Battle Object
• Field Attributes (weather, other effects like trick room)
Player Side Attributes
• Side Attributes (entry hazards, protective effects like light screen)
Opponent Side Attributes
• Side Attributes (entry hazards, protective effects like light screen)
Battle Pokemon
Battle Pokemon
Fixed Attributes
• Species
• Moves
• Item
• Ability
• Level
Variable Attributes
• Health
• Status Data (EX: burn, paralyze)
• Volatile Status Data (EX: confusion)
• Boost Data (EX: increase in attack)
• Currently Active

In this model, s_{start} can simply represent the current state of the battle - we do not need to consider previous moves since this is already encoded in the state. At each state, $Actions(s)$ generally includes all of the moves the active Pokemon has available, as well as all other alive Pokemon that we can switch to. In some cases, moves are disabled, we cannot switch, or we are forced to switch, reducing the size of $Action(s)$. Some Pokemon are capable of Mega Evolving, a feature in which a Pokemon can switch to a more powerful form while simultaneously making a move. However, we make the simplifying assumption that we always mega evolve when possible, as Mega-Evolution is almost always beneficial; this significantly decreases the branching factor. $Succ(s, a)$ will simply be the result of the simulator applying action a of the current player.

Terminal states are states in which either player has no alive Pokemon left. We delay all rewards until these terminal states: $Utility(s) = \infty$ if all opposing Pokemon have no health, and $Utility(s) = -\infty$ if all of our Pokemon have no health. We do not consider the rare occurrences of ties in Pokemon battles.

$Player(s)$ is determined by the simulator - in cases where both players have to make a move, the maximizing agent will pick an action first, and then the minimizing agent will pick an action.

Special cases will be handled by the simulator. For example, when a player's Pokemon faints, the player will need to move twice in a row to select a new Pokemon and make a new move.

In this model, we want to pick the correct sequence of moves that will maximize our end-game utility. Thus, our goal is to reach a state in which all opposing pokemon have zero health.

4 Challenges

4.1 Algorithmic Challenges

The primary challenge in writing a Pokemon AI is dealing with the massive state space. On each turn, a player can either choose one of four Pokemon moves or switch to one of five Pokemon. By just using Minimax search to determine all possible states at a depth of 2, there are 6561 possible ending states. Given that this bot will be used in an online setting, the bot should be able to determine actions within a few seconds to maintain opponents' interest in battling.

4.2 State Challenges

Even though Pokemon Showdown is a turn-based game, keeping track of state is challenging. The state includes global field features, features on each player's side, and information about all six Pokemon. Furthermore, a large portion of this information is hidden at the start, and only revealed with time. For example, the opponent's Pokemon are unknown until they are sent out to battle, and an opposing Pokemon's moves are unknown until it uses them. We can assume that the opponent's unrevealed Pokemon are non-existent for simplicity. In addition, there are several hidden factors that are difficult to detect. For example, items called choice items can increase the attack power of Pokemon while restricting that Pokemon to a single move. It can be difficult or impossible to detect whether an opposing Pokemon is carrying a choice item, but these items can significantly affect the utility of opposing Pokemon.

5 Method

5.1 Baseline Algorithm

In order to gauge the performance of the Minimax search, we compare it to a baseline algorithm. The baseline algorithm uses domain specific knowledge to rank moves in any given situation, choosing moves with the highest rankings. The baseline only takes into account the current state and not future states. In addition, the domain knowledge for the baseline algorithm has game mechanics hard-coded into the algorithm. Thus, while the majority of game mechanics are considered in the baseline algorithm, certain moves and abilities are not accounted for, which can decrease performance.

5.2 Minimax Algorithm

To approach these challenges, we used the Minimax search algorithm to determine the cost-minimizing moves for the bot. Minimax is a good approach in Pokemon battling since games are two-player and zero-sum. Several optimizations were necessary to traverse the large state space of the battles.

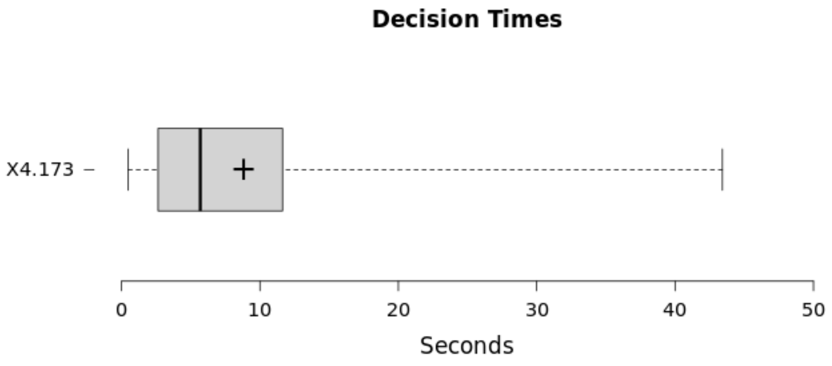


Figure 1: Distribution of 50 decision times

5.3 Alpha-Beta Pruning and Move Ordering

We implemented alpha-beta pruning to reduce the number of nodes searched by Minimax. The process essentially rules out nodes that are sub-optimal compared to previously examined nodes.

Our baseline algorithm uses domain specific knowledge to rank moves, picking the moves with the highest rankings. We use this ranking as a heuristic to implement move ordering, thus maximizing the potential of alpha-beta pruning. For example, we may try super effective moves and status-inducing moves first before trying not very effective moves, quickly pruning the tree.

These optimizations allow the bot to run Minimax search to a depth of 2 with a median decision time of 5.5135 seconds. However, we see a considerable long tail of times that range beyond 40 seconds. This likely occurs in cases where there are no obviously good moves and neither Pokemon on the field has a significant advantage over the other. These decision times are well under the 150 seconds allocated for each player to make a move. The distribution of times can be seen in Figure 1.

5.4 Evaluation Function

Because the Minimax search is run to only a depth of 2, a strong evaluation function is necessary to precisely evaluate states. The evaluation function takes into account factors that are directly stored in the state. The evaluation function also considers other factors such as whether the current Pokemon can deal super effective damage and whether the current Pokemon is faster than the opposing Pokemon. To weight the factors, we borrowed some of the weights utilized in the Technical Machine evaluation function and manually tweaked the weights after observing the bot's behavior in battle. The final weights can be seen at github.com/rameshvarun/showdownbot/blob/master/weights.js.

6 Implementation Details

Pokemon Showdown does not have a bot framework. However, Showdown is open-source, meaning that we can adapt code from both the client and the server. To do this, we pored over both GitHub repositories, studying the protocol, re-implementing certain components on our side, and copying helper functions as necessary. Our bot pretends to be a client, and connects to the Pokemon

Showdown server via websockets. It makes appropriate POST requests to specific PHP endpoints in order to authenticate itself, loading account information from a secret JSON file.

6.1 Communication Protocol

On each turn, the Pokemon Showdown server sends clients information about the battle and possible moves that a player can take, in the form of a JSON object. This JSON object either encodes a move request (player can move or switch), a forceSwitch request (player must pick a Pokemon to switch to), or a wait request (player must wait for opponent to select a move). Each player sends his selected action in the form of a formatted choice string, which looks like the following. [1]

```
--Search for a random battle--
/search randombattle

--Select the move Fire Blast--
/choose move fireblast

--Switch to the Pokemon at position 3--
/choose switch 3
```

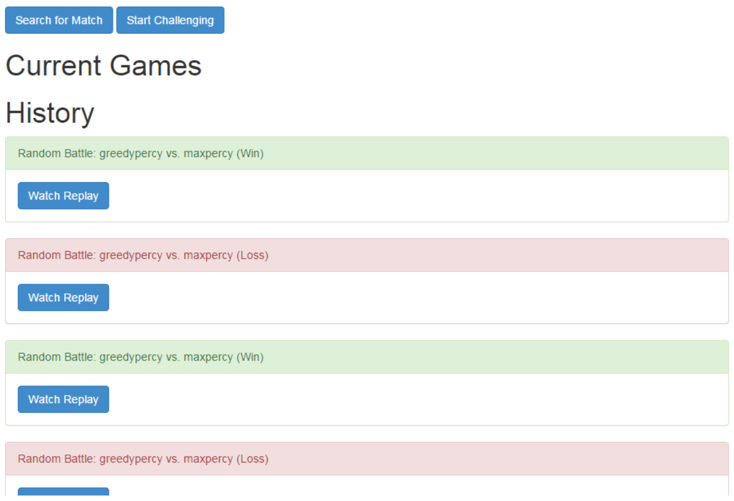
When both players have taken their action, the server begins a simulation of the battle. The events that occur in this simulation are then sent to the client in a battle log format.

```
|move|p2a: Bronzong|Earthquake|p1a: Magmortar
|-supereffective|p1a: Magmortar
|-damage|p1a: Magmortar|42/100

|move|p1a: Magmortar|Fire Blast|p2a: Bronzong
|-supereffective|p2a: Bronzong
|-damage|p2a: Bronzong|0 fnt

|faint|p2a: Bronzong
```

In this example, the opponent's Bronzong (player 2) uses the move earthquake, which deals super effective damage against the bot's Magmortar (player 1). The bot's Magmortar then uses Fire Blast, which deals super effective damage against the opponent's Bronzong and causes it to faint.

The code that handles all communication with the server and information parsing can be found in battleroom.js. The BattleRoom object then invokes the decide function of the respective algorithm.

[1]Note that a client can participate in many battles at once, which are all multiplexed throught he same WebSocket. This is handled through a room system. A message is sent to the client from a specific "room", and a client returns a response to a specific "room". Some commands, such as searching for a battle, are global and are not associated with a particular room.



Random Battle: greedypercy vs. maxpercy (Win)

Random Battle: greedypercy vs. maxpercy (Loss)

Random Battle: greedypercy vs. maxpercy (Win)

Random Battle: greedypercy vs. maxpercy (Loss)

Figure 2: The dashboard of the web console.

6.2 Web Console

In order to administer our bot remotely, we created a web console, shown in Figure 2. This console has buttons that allow us to search for a game on the official ladder, as well as to start challenging - in this mode the bot continually plays games until we explicitly stop it. The console will also show in-progress games, with a link to either spectate (on Pokemon Showdown itself) or look at the minimax tree. Finally, the console maintains a history of all games played, with a link to watch a replay. We use NeDB, an embedded database for Node.js, to store our game data. This includes the replay, whether or not we won the match, and any decision information, such as the minimax tree.

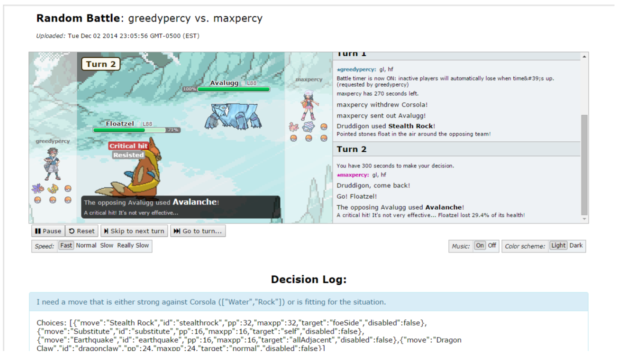
Due to issues with using Pokemon Showdown's replay system, we were forced to save replays locally if we wanted to watch previous battles. To do this, we simply save the battle log (the system used to communicate the outcomes of a battle). In order to play it back, we repurposed Showdown's replay system and inject it with our own battle logs.

For detailed debugging, we developed a visualization of the minimax tree that we used to inspect the bot's decisions. Each node in the tree is visualized as a collapsible panel. At each level, we can see the minimax value, the type of node, as well as various details about the state. At the leaves, we can see the result of the feature extractor. Green panels are used to represent the expected Minimax actions taken by the bot and the opponent.

6.3 Battle Simulation / Minimax Search

The Pokemon battle system is incredibly complex - every move is weighted by a multitude of factors, from priority and weather to stats and status effects. The battle system is also filled with many special cases. To create our own battle simulation from scratch would simply be intractable. Instead, we tried to adapt the simulator code used by Showdown itself. This leads to several challenges.

First, this eliminates the possibility for an Expectiminimax tree. Randomness is prevalent in Pokemon battles, and reconfiguring the battle simulator to take into account probability would



Random Battle: greedypercy vs. maxpercy

Uploaded: Tue Dec 02 2014 23:05:56 GMT-0500 (EST)

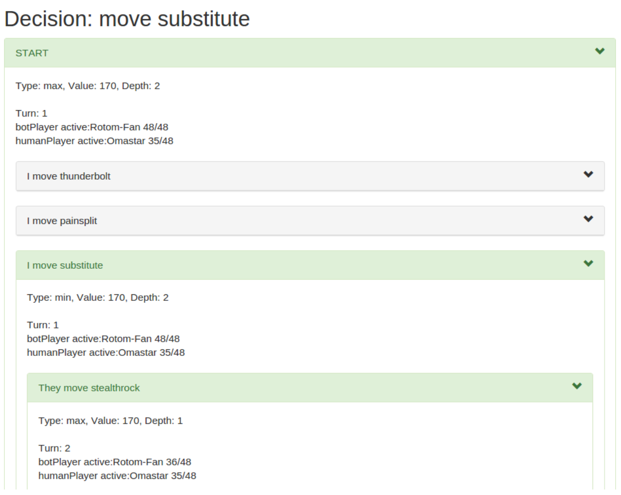
```
Turn 1
★greedypercy: gl, hf
Battle timer is now ON: inactive players will automatically lose when time&#39;s up.
(requested by greedypercy)
maxpercy has 270 seconds left.
maxpercy withdrew Corsola!
maxpercy sent out Avalugg!
Druddigon used Stealth Rock!
Pointed stones float in the air around the opposing team!
Turn 2
You have 300 seconds to make your decision.
★maxpercy: gl, hf
Druddigon, come back!
Go! Floatzel!
The opposing Avalugg used Avalanche!
A critical hit! It's not very effective... Floatzel lost 29.4% of its health!
```

Decision Log:

I need a move that is either strong against Corsola (["Water","Rock"]) or is fitting for the situation.

```
Choices: [{"move":"Stealth Rock","id":"stealthrock","pp":32,"maxpp":32,"target":"foeSide","disabled":false},
{"move":"Substitute","id":"substitute","pp":16,"maxpp":16,"target":"self","disabled":false},
{"move":"Earthquake","id":"earthquake","pp":16,"maxpp":16,"target":"allAdjacent","disabled":false},{"move":"Dragon
Claw","id":"dragonclaw","pp":24,"maxpp":24,"target":"normal","disabled":false}]
```

Figure 3: A replay of a stored battle.



Decision: move substitute

START

Type: max, Value: 170, Depth: 2

Turn: 1
botPlayer active:Rotom-Fan 48/48
humanPlayer active:Omastar 35/48

I move thunderbolt

I move painsplit

I move substitute

Type: min, Value: 170, Depth: 2

Turn: 1
botPlayer active:Rotom-Fan 48/48
humanPlayer active:Omastar 35/48

They move stealthrock

Type: max, Value: 170, Depth: 1

Turn: 2
botPlayer active:Rotom-Fan 36/48
humanPlayer active:Omastar 35/48

Figure 4: The minimax tree.

require rewriting many portions of the battle logic code. Second, Showdown's battle system expects full state knowledge, which, as mentioned before, is not available. To fix this, we took some shortcuts. First, we assume that unknown opponent Pokemon are non-existent, and assume that the opponent can only use the Pokemon that the bot has seen so far. Thus, unknown opponent Pokemon are excluded from the minimax tree. Second, we don't know the opponent's moves from the start. However, we do know that a Pokemon's moves are picked by selecting 4 random moves from a set of on average 7 moves per Pokemon. Hence, we assume that opposing Pokemon have access to all 7 moves until we observe the four distinct moves that an opposing Pokemon actually possesses in battle.

The minimax search itself is straightforward; however, it requires that we are able to deep copy our game state, in order to simulate many possibilities. The game state is filled with circular references (eg. each pokemon has a reference to the battle itself), so shallow cloning is not sufficient. We started with a general-purpose deep/circular cloning library. However, significant changes had to be made, as the library's run-time is $\Theta(n^2)$, with n Javascript objects. The library also flattened prototypes, which is unnecessary for our use case. We forked the library and removed the flattening of prototypes; other optimizations allowed us to reduce the expected runtime to $\Theta(n)$. This decreased clone times from upwards of 20 seconds to milliseconds. Still, cloning is a large bottleneck in our minimax search, forcing us to cut off searches at depth 2.

7 Testing

To test the performance of Percymon, we allowed the bot to play 134 battles on the ranked Pokemon Showdown Random Battle ladder. In addition, we developed a secondary bot that used the baseline greedy algorithm as mentioned earlier to select moves. We allowed this greedy bot to play battles during the same period, and compared the performances of the bots to a human player, which served as an oracle algorithm.

8 Results

Algorithm (Account Name)	Elo	GXE	Glicko-1	Wins	Losses
Minimax (ThinkingPercy)	1270	55	1540 ± 31	71	63
Baseline (GreedyPercy)	1048	41	1431 ± 25	107	142
Oracle (MongolMouse)	1724	83	1793 ± 42	55	18

Pokemon Showdown provides several metrics for measuring player ranking. Elo is a rating system, originally designed for Chess, that is in use in a variety of competitive games. This is Showdown's primary ranking system and is used for sorting its final ladder. The Glicko-1 rating system was devised as an improvement to Elo and takes into account ratings deviation, which is shown by the $\pm$ term. GXE stands for Glicko X-Act Estimate, which is the chance that a player has of winning against any other random player [3]. GXE is derived from the Glicko-1 rating.

9 Analysis

At first, it is tempting to look at the win rate to evaluate the performance of an algorithm. However, this is only effective if match-ups are random. In fact, players are matched against those with similar ratings. Under this system, the win rate is supposed to converge on 0.5, and both bots do come close to that. However, even with this constraint, the Minimax bot was able to achieve a win rate of 52.98%, higher than the 42.97% win rate of the baseline algorithm.

To better gauge performance, we can look at the various ratings systems. In each of these systems, the Minimax algorithm bot achieved a higher ranking than the baseline algorithm bot, with a 222 point increase in Elo, and a 109 point increase in Glicko-1 rating. The 14 percentage point increase in GXE suggests that the minimax bot is more likely than the baseline to win against a random player.

However, we see that there is still a huge gap between the bots and the oracle (human) player, who is able to play at a much higher level. The human player has a 454 point elo advantage, and a 253 point Glicko-1 advantage. His high GXE means that he is extremely likely to beat a random other player, while the minimax bot will only beat about half of its opponents.

10 Conclusion

In conclusion, we see that the Minimax algorithm is a significant improvement over a domain-specific baseline algorithm. In sifting through replays, we see that there is a wide range of strategies that the Minimax bot can take advantage of. One example is the practice of using moves that increase a Pokemon's stats. In one instance, the Minimax bot's active Pokemon can defeat the opposing Pokemon in four hits. However, instead of using an attacking move repeatedly, the bot can use a boosting move once, allowing the bot to defeat the opposing Pokemon in three hits instead. In addition, the bot will take advantage of unique moves and abilities, such as Pain Split and Magic Guard, to defeat opponents. Such behavior is difficult to encode within the greedy algorithm.

The bot still makes many mistakes in trying to optimize the evaluation function. In one common example, the bot will have a Pokemon with low HP on the field which cannot make any more significant contributions to the battle. However, since there is a large positive weight on the number of alive Pokemon, the bot will attempt to save this Pokemon by switching to a healthy

Pokemon and letting it take a hit. As a result, the new active Pokemon is less useful to the bot since it may lose a significant amount of HP. Further adjustment of weights is necessary to eliminate behavior errors like this.

11 Future Work

A number of improvements can be made in order to increase the performance of the algorithm. The first improvement that we can make is to implement TD Learning. We had attempted to implement TD learning to find optimal weights for features in the evaluation function. However, TD learning did not produce reasonable results in the given time-frame; for example, beneficial attributes, such as player Pokemon HP, were assigned negative weights. In the current iteration of Percymon, we were not able to run enough iterations of TD Learning because extra time was needed to connect to and communicate with the Pokemon Showdown server. Running a local server to run iterations of TD Learning could potentially speed up this process, allowing us to obtain reasonable weights for the evaluation function.

Another improvement that we may consider is to implement prediction of the opponents' moves. Currently, the bot assumes the worst case scenario for each state. However, assuming the worst case in the minimax algorithm can be sub-optimal, as the opponent may consider it too risky to select moves leading to this worst case. Implementing prediction would allow Percymon to pick moves based on what actions the opponent is mostly likely to choose as opposed to opponent actions that would lead to the worst situation for Percymon. As an example of prediction, suppose that the bot's active Pokemon is fire type, while the opponent's active Pokemon is grass type. In addition, suppose that the bot has observed that the opponent has a tendency to switch to a water-type Pokemon in situations like this one. A future iteration of Percymon may predict this switch and select an action that is strong against water-type Pokemon, such as a grass type move.

One interesting idea would be to combine the battling AI with a CSP solver that builds good teams, allowing the bot to play regular ranked battles. In addition, this system can be be paired with a team prediction system. There are many commonly used move sets and team combinations, so a Bayesian network could potentially be used to guess an opponent's Pokemon and moves, given limited observations.

References

[1] "User Stats - Pokemon Showdown!" User Stats - Pokemon Showdown! Pokemon Showdown, n.d. Web. 12 Dec. 2014.

[2] Stone, David. "Technical Machine." Technical Machine. N.p., n.d. Web. 12 Dec. 2014.

[3] Clyro, Biffy. "GLIXARE: A Much Better Way Of Estimating A Player's Overall Rating Than Shoddy's CRE." Smogon University. N.p., 17 Feb. 2006. Web. 12 Dec. 2014.