Showdown AI Competition

Scott Lee
Tandon School of Engineering
New York University
Brooklyn, New York 11201
Email: sl3998@nyu.edu

Julian Togelius
Tandon School of Engineering
New York University
Brooklyn, New York 11201
Email: julian@togelius.com

***Abstract*—We present the Showdown AI Competition, a game-based AI competition built around a clone of the popular game Pokemon. This is a game of turn-based team battle, where the objective is to defeat an opponent team using clever combinations of creatures and their abilities. The gameplay is reminiscent of computer role-playing game battles and collectible card games. The game has characteristics, such as the combination of turn-based gameplay and partial observability, that are unusual in current game-based AI competitions and therefore offers a fresh challenge.**

I. Introduction

Games have historically demonstrated that they are ideal testbeds for artificial intelligence algorithms. They provide an explicit set of rules and a clearly defined system of cause and effect, which allows researchers to define the nature and complexity of the problems in play. The ability to formulate these rules and systems grants the freedom to define and tackle a wide variety of problems, from single player platformer games such as Super Mario Bros [1] to adversarial strategy games [2].

In academia, research into AI for games is particularly useful because games can be selected or configured to mirror problems in other domains. As such, solutions and high-performing agents in these games can be applied to overcome challenges in other areas. For example, Monte Carlo Tree Search, which has found a great deal of use in playing the board game *Go* [3], has also been utilized to solve scheduling problems [4]. The ability to treat games as a surrogate for other problems has also given us an ability to test solutions to dangerous problems in a safer way. Rosser et al. demonstrated that video games such as Super Monkey Ball correlated with surgical skills and presents a possible way to integrate video games into training curricula for surgeons [5]. Similarly, games can be used to test the efficacy of AI algorithms in high-risk domains such as self-driving cars without endangering property or lives.

There is also commercial incentive for the development of game AI. As artificial intelligence often finds usage in video games as enemy units and content generators, improvement of these algorithms translates into a change in gameplay experience for players. Higher performing agents can provide more challenging opponents, and a wider variety of agents can capture a larger number of playstyles. There are numerous roles for AI in games, including powering non-player characters in various roles [6] but also for e.g. game adaptation, procedural content generation, and design assistance [7]. Expanding a game developer's toolkit is likely to increase variety in NPC behavior in games, and potentially allow for a greater degree of personalization for player skill.

Game-based AI competitions are a useful way to promote AI research into a specific domain while also providing a means to evaluate agents against one another [8]. These competitions typically involve the development of a game framework and API, which competitors then use to develop algorithms to play the game in question. Upon submission, these agents are ranked based on some metric or rule. For single player games agents can be ranked by some scoring metric while multi player games can make use of round robin or single elimination tournaments. Competitions are useful to developers as this eliminates the often burdensome step of developing entire frameworks to test AI algorithms. They provide a standardized metric against which agents can be compared, making it possible for different groups of researchers and developers to directly compare algorithms against one another.

A number of competitions are run every year at the IEEE CIG conference, and also regularly at other conferences such as IJCAI and AIIDE. A large number of papers published at these and other venues are based on work undertaken for these competitions or use the competitions' various software frameworks. A list of the competitions that have had significant impact in this sense should include at least the Ms Pac-Man Competition [9], the StarCraft Competition [2], the Mario AI Competition [1], the Simulated Car Racing Championship [10], and the General Video Game AI Competition [11]; new competitions are proposed every year, with some having more staying power and impact than others. (Additionally, there are industry-sponsored game-based AI environments, such Microsoft's *Project Malmo* and OpenAI's *Universe*, but these are typically not configured as competitions.)

When proposing a new competition, it is important that it offers a challenge that is not found—or not found in this form—in existing competitions. Seeing that the existing CIG competitions are composed of mostly real-time video games with mostly perfect or locally perfect information, it can be argued that there is a need for competitions that explore other parts of the game design space, while still being based on modern video games. In particular, the type of adversarial

978-1-5386-3233-8/17/$31.00 ©2017 IEEE

battles that is often found in collectible card games and computer role-playing game combat is not represented in existing game-based AI competitions. A good example of this is the very popular *Pokemon* series of games, which are heavily based on such of battles.

This paper outlines the rules and technical specifications of the Showdown AI Competition, which requires competitors to create agents that participate in the combat system of the Pokemon series of games. First we describe the general rules of Pokemon battling. We then outline the rules of the Showdown AI Competition, after which we present the Showdown AI Framework, its technical specifications, and the resources it offers AI developers. We present the challenges Pokemon offers as an AI benchmark, as well as strategic considerations that Pokemon offers as a game. Finally, we present sample controllers and their comparative performance in Pokemon battling.

II. Pokemon

Pokemon is a game franchise that was developed by Game Freak and published by Nintendo in 1996 [12]. It has since spawned a number of sequels as well as a number of spin-off games which span several genres. The main series consists of role-playing games in which the protagonist travels across a region, capturing creatures called Pokemon, and training them for use in battle. We will here primarily discuss on the battle aspect of the main games, as that is what the competition is about. At the time of writing, there have been seven distinct generations of Pokemon games, with a number of changes occurring between each, but the underlying formula remains unchanged. In battle, two teams of up to six Pokemon battle one another until all of the Pokemon on one team are incapacitated.

Since the original release of Pokemon Red and Blue[13], there have been six additional generations of games, each of which has added systems and complexity to the combat system. Generation 2 added two additional stats for damage calculation, generation 3 gave each Pokemon an innate passive ability, generation 4 redefined which stats were used in move damage calculation, and so on. Each of these generations had a substantial impact on competitive strategy, which was investigated and codified by the game's competitive community[1]. Because new Pokemon games are released on a regular basis, the game's strategic landscape is highly fluid.

As much as the game has changed since 1995, the win condition has remained unchanged. The goal of Pokemons combat is to defeat all of the opponents Pokemon. Before the beginning of a battle, both players construct or are given teams of six Pokemon, each of which have up to four moves. All Pokemon have a value known as hit points (hp), which depletes as a Pokemon takes damage from attacks. When this value reaches 0, the Pokemon is considered incapacitated, and the game ends when all of a players Pokemon reach this state.

[1]Smogon is one of the game's largest competitive community sites, and can be found at www.smogon.com

Pokemon's combat system uses an atomic turn system, meaning that rather than alternating turns and choices between players as in games such as chess, Pokemon receives decisions from both players, and then resolves both decisions simultaneously. In a single turn, a player can choose to use one of their active Pokemon's moves or switch to another Pokemon. Once both players have chosen a move, the game uses a priority system to resolve the events of the turn. Except in very specific cases, switching takes precedence over moves, then moves with special priority are processed. If both players have chosen moves without special priority, the Pokemon with the higher Speed stat moves first, and the other Pokemon moves second. It is worth noting that Pokemon can be incapacitated before their move occurs, in which case the player's turn is effectively skipped.

III. Competition Rules

A. Pokemon Rules

The Showdown AI Competition includes the Pokemon, moves, and rules of generation 6 Pokemon games: Pokemon X, Y, Omega Ruby, Alpha Sapphire (XY ORAS). Additionally, the Showdown AI Competition uses a set of rules defined by Smogon, one of Pokemon's largest competitive communites. These rules are[14]:

- Species Clause - No team can have two of the same Pokemon. Pokemon with different formes are still considered the same Pokemon.
- Sleep Clause - A player is disallowed from inflicting sleep on more than one of the opponent's Pokemon at a time.
- Endless Battle Clause - Players are disallowed from intentionally preventing a battle from ending.
- Baton Pass Clause - Only one Pokemon can have Baton Pass. Also, no Pokemon can have Baton Pass, a speed boosting ability, and a stat boosting move at the same time.
- Move Bans - The following moves are banned:
  - Double Team
  - Minimize
  - Fissure
  - Guillotine
  - Horn Drill
  - Sheer Cold
  - Swagger
- Ability Bans - Moody and Shadow Tag are banned
- Item Bans - The following items are banned:
  - Gengarite
  - Kangaskhanite
  - Lucarionite
  - Mawilite
  - Sablenite
  - Salamencite
  - Soul Dew

Additionally, a number of Pokemon are banned from use in this competition. Using Smogon's generation 6 tier list, all Pokemon in or above the Ubers tier and in or below

the NeverUsed tier[15] are banned from this competition. This is designed to maximize the chance of balanced battles between agents. Additionally, the Pokemon Zoroark has been banned from the Showdown AI Competition due to a uniquely complex challenge it poses to AI opponents (more on this below).

B. Showdown AI Competition Rules

One of the goals of this competition is to create a testbed that is as faithful to the original Pokemon games as possible. However, several changes and additional rules have been made specifically for the competition.

- Once a move request is made to an agent, it has 20 seconds to return a valid move. Failure to do so will be treated as a pass, and that player's turn will be skipped for that turn.
- If a battle lasts longer than 500 turns, the game is ended with a result of "No Contest".
- If an agent throws an error or exception, that is considered a forfeit and its opponent will be considered the winner.

Battles in the Showdown AI Competition are operated as one-on-one single battle with no team preview. Teams are randomly populated with six Pokemon that conform to the rules outlined in Section III-A. Additionally, all Pokemon movesets are randomly generated using moves that are considered viable in competitive play. That is, moves such as Thundershock and Splash are guaranteed to be excluded from random selection, as they are considered either outclassed or useless.

Once each player has been given a randomly generated team, they battle against one another three times. The teams are then swapped between the players and three more battles occur. This six battle format reduces the possibility that an agent wins games because the randomly generated teams are unbalanced. The reasoning behind this is the assumption that if an agent is substantially more capable than its opponent, it should be able to win battles with both teams.

The competition consists of three rounds: a qualifying round, a preliminary round, and a final round. The qualifying round pits all agents against the One Turn Lookahead sample agent described in Section VII for 15 sets of six battles each. Those agents that win the most games against the sample agent proceed to the preliminary round, which is operated as a round robin tournament. (The size of this tournament will be decided at a later date.) Each agent plays against all other agents for five sets of six battles each. The four players with the best win-loss ratio will participate in a double-elimination style tournament. The agents will be randomly matched, at which point each pair of agents will play 15 sets of six battles each. The final ranking is determined by agent performance in the double-elimination final round.

IV. The Showdown Framework

The Showdown AI Competition uses a modified version of Pokemon Showdown[16], an open-source Pokemon battle simulator written in Node.js and developed by Guangcong Luo[17]. The framework operates by preserving the game engine from Pokemon Showdown and supplying an interface that emulates the socket-based interface of the Pokemon Showdown server. The framework as a whole contains four main layers: the game layer which houses the shared canonical game state, the communication layer which facilitates communication between the game engine and other objects, the interface layer which parses information coming in from the communication layer and updates the personal game state, and the agent layer, which makes decisions to be sent to the canonical game state for playing. Each player has a personal game state, which replicates as much of the game state as possible given the information a single player would have access to. This is different from the canonical game state, because a personal game state has no way of disambiguating hidden information, and is thus treated as a default null value until it is revealed to the player. The agent will primarily interact with objects of the following classes:

- *Battle* - Represents a game state. Contains information about field effects and references to *BattleSide* objects.
- *BattleSide* - Represents a player. Contains information about side effects and references to *BattlePokemon* objects.
- *BattlePokemon* - Represents a single Pokemon. Contains information about the Pokemon's current status.

The code for each of these classes can be found in *battle-engine.js*.

A. Reference API

The Showdown framework supplies agents with a set of functions that retrieve data about various aspects of the game. These reference functions are contained in *tools.js* and can be accessed by referencing the global *Tools* object. The following functions are available:

- *Tools.getMove(moveid)* - Returns information about a move with id *moveid*. The format of a returned move can be found in *moves.js*. Relevant fields in the return object include *move.basePower*, *move.type*, *move.status*.
- *Tools.getTemplate(species)* - Returns information about a Pokemon species. The format of a returned template can be found in *pokemon.js*. Relevant fields in the return object include *template.baseStats*, *template.types*, and *template.abilities*.
- *Tools.getEffectiveness(source, target)* - Returns an integer value representing the effectiveness of an attack *source* against a *target*. A value of 1 represents super effectiveness, -1 represents resistance, and 0 represents neutrality. This function does not account immunity.
- *Tools.getImmunity(source, target)* - Returns a boolean value representing a *target's* immunity against a *source* attack. A value of false represents immunity.

Details regarding syntax and arguments for the invocation of these functions can be found in documentation packaged with the framework.

B. Forward Model

The *Battle* class provides methods of predicting outcomes of moves. The class's *getDamage(user, target, move)* function estimates the damage *move* would do if used by *user* on *target*. It is important to note that this function takes into account the state of the game to run its calculations, meaning which game state is used to call this function can affect the outcome. Additionally, because Pokemon's damage calculation involves a degree of stochasticity, and because *getDamage* uses Pokemon's damage calculation formula as-is to make its prediction, the result of these calculations is an estimate rather than an exact prediction.

The *Battle* class's *choose(player, choice)* method allows agents to simulate full turns. Because turns in Pokemon are processed atomically, turn simulation can only start once both players have made a decision, meaning *choose* must be called twice (once for each player) to advance the forward model. The choice is a string which can be formatted as follows:

- "move X" - This indicates the decision to have the current active Pokemon use move X.
- "switch X" - This indicates the decision to switch to the Pokemon at position X in the team.
- "forceskip" - This indicates the decision to skip the current turn.

Once both players have made a decision, the game state advances automatically, simulating all events that take place until the next request for a move. It is important to note that due to the stochasticity and hidden information present in the game, this simulation is an estimation of the resultant game state.

V. Challenges and Considerations

Pokemon and the Showdown AI Competition can usefully be compared and contrasted to other games from the point of characteristics that affect AI gameplay as follows:

A. Branching Factor

Tree search agents will be interested in the number of results that could emerge from a single state. Ignoring additional complications from stochasticity, this largely reduces to the number of choices a player can make at a given time. In Pokemon, this number varies based on the state. In a clean state without any effects or modifying factors, the branching factor can be expected to be between four and nine: four possible moves, five possible switches. In general, this is the most common scenario in-game, and as such, nine is a reasonable assessment of the game's average branching factor. Moves such as Encore and Fire Spin can reduce this branching factor for their target, as these effects lock them out of several options. Conversely, some moves require additional choices after being used. For example, Baton Pass forces the user to switch Pokemon after being used, and so Baton Pass as an option can be split into five separate options, one for each Pokemon. As such, the maximum theoretical branching factor is 25: four moves leading to switches to five Pokemon plus five switches.

B. Turn Count and Infinite Looping

Perhaps one of the first considerations when designing an algorithm for any challenge is understanding how far the agent may have to predict into the future. In human play, the average battle ends before turn 60, but it is important to note that there is no theoretical upper limit. Although the Endless Battle Clause outlined in Section III-A provides some protection against infinite loops, it fails to account for voluntary looping. In the event both players switch Pokemon every turn, the game can go on without ending. This may occur if both agents recognize that choosing to switch in the current situation is optimal, only to find that the opponent has also switched Pokemon, making switching back to the original Pokemon the optimal move. In human play, these situations should almost never occur, but Agents may not be be designed with the ability to identify and break this cycle. It is worth noting that infinite switching does not violate the Endless Battle Clause in Section III-A, as neither player is intentionally locking the other into into an infinite loop. Both players are at fault in this scenario. The turn limit rule outlined in Section III is designed to counter this possibility.

C. Turn Atomicity

Although Pokemon is categorized as a turn-based game, it handles turns somewhat differently from other turn-based games such as Chess or Hearthstone. Rather than processing decisions separately in alternating order, Pokemon processes both players' decisions simultaneously. This means that the game state can change between the time the player makes a decision and the time that decision is enacted. For example, at the beginning of the turn, player 1 decides to attack. Once both players have made their decisions, it is revealed that player 2's Pokemon has a higher Speed stat, and thus moves first. Player 2's attack incapacitates player 1's Pokemon, canceling their move. Because these scenarios can occur, agents may find themselves making decisions based on game states that are already outdated.

D. Categorical Dimensions

Comparison of health values is a valid way to evaluate the favorability of states in Pokemon. However, this metric only captures a small portion of the full situation. Many moves in the game forgo damage for some other effect, ranging from modifying stats to applying damaging status effects to the opponent. Unlike values such as health and stats, which can be measured as-is, status, side, and field effects are often categorical. For example, it is difficult to quantify the value of the burn and paralysis status effects, which makes comparing the two difficult. This is further compounded by the fact that several of these categories have a large number of potential values, and are not mutually exclusive. A single Pokemon can be under the influence of multiple effects simultaneously, and even if each effect can be individually quantified, it raises questions of how to compound effects that may or may not be related, interconnected, or amplified by one another.

E. Stochasticity

An important consideration in agent design is whether or not the game or challenge acts as a deterministic system. A fully deterministic game means that it is possible to predict the exact outcome of any action or decision made in a game, and as such, any predictions made regarding outcomes can be considered accurate and reliable. In contrast, in a game with stochasticity any predictions made would be predicated on some assumption about the outcome of a random variable. The impact of randomness in Pokemon is significant, and much of the games competitive strategy comes from being able to accurately estimate stochastic events.

This stochasticity exists on a number of levels, which makes accurate evaluation of options difficult. For example, damage done by a move is highly stochastic. The damage calculation formula for attacks includes a random multiplier between the values of 0.85 and 1.0. Additionally, all attacking moves have an innate 6.25 percent chance of being a critical hit, which increases the damage dealt by 50 percent.

It is also worth noting that this randomness can also cause moves to fail altogether. All moves in Pokemon have an accuracy value between 0 and 100. The represents the percentage chance of a move succeeding. This is especially important for attacks such as DynamicPunch and Zap Cannon, which have exceedingly high destructive potential, but a low chance of success. An appropriate valuation of accuracy is also important for understanding the value of strategies that modify the odds of success. For example, weather effects have historically been a key part of Pokemons metagame, and some of the value they bring is derived from their ability to raise and lower accuracy values. A notable example is rain's effect of increasing Thunder's accuracy from 70 to 100. Conversely, there are ways to decrease an opponents odds of success. For example, while hail is active, the Snow Cloak ability applies a 20 percent penalty to accuracy for all attacks targeting it.

There is also the consideration of secondary effects in moves. Many attacks in Pokemon have a chance of applying additional effects after connecting. These effects are often significant and lend great value to these moves. Notable examples are Scald, which has a 30 percent chance of inflicting the burn status, and Air Slash, which has a 30 percent chance of flinching the opponent (forcing them to skip their turn).

F. Hidden Information

Hidden information exists primarily on two levels in Pokemon. At the Pokemon level, an opponent Pokemons moves, ability, and stats are hidden from the opponent. A Pokemon's ability, while impactful, can only be one of up to three values, and for many Pokemon, they have only one right answer ability, and so this issue can be largely eliminated by applying domain knowledge. A Pokemons stats can vary greatly based on hidden information values, but the application of knowledge regarding cookie-cutter builds for Pokemon can greatly mitigate this issue. Moves, however, can be problematic to predict. Although a Pokemon can only enter battle with up to four moves, those four moves can be any move in a Pokemons learnset, which can be large. As an example, Pikachu has a learnset of over 100 moves. Domain knowledge is again useful here, as it can be used to weed out moves that are largely outclassed or considered non-viable in competitive play. However, even after narrowing the possibilities, it is still unlikely that an agent will be able to accurately predict all four of an opponent's moves.

Hidden information also exists at the team level. In some battle formats, players are not informed as to which Pokemon the opponent has in their team. This introduces a great deal of difficulty when it comes to predicting when and how an opponent will switch their Pokemon. This also presents some issues for simulation. It is comparatively easy to simulate far enough ahead to defeat the current opposing Pokemon, but without any way of knowing what the next Pokemon may be, it is difficult to formulate a plan that will hold for the remainder of the battle. For formats wherein players build their own teams, domain knowledge can be applied to identify synergies between Pokemon to narrow the possibilities, but in formats where teams are randomly generated, there are no such cues to leverage. This presents a challenge in random-based formats wherein accurate prediction past a certain point is almost impossible, simply due to the vast number of possibilities in play.

G. Deception

There is a very specific scenario in which a players knowledge of the game state is simply incorrect. Zoroark possesses a unique ability known as Illusion, which causes it to be disguised as another Pokemon in battle. This creates a special type of persistent hidden information involving the possibility that any Pokemon the opponent uses may in fact be a Zoroark in disguise, which is especially problematic because Zoroarks typing gives it an immunity to Psychic type attacks. This means that a player under the impression that a Psychic type attack would be effective would attempt to use this attack and fail. If an agent has no means of acknowledging the fact that it failed or attributing this failure to the possibility that the opponent is secretly a Zoroark, the player is liable to attempt to repeat this attack with similarly futile results. It is however difficult to make decisions while constantly accounting for the possibility that the opposing Pokemon is disguised, as that creates a great deal of uncertainty in strategic decision making. For the time being, Zoroark is the only Pokemon in standard competitive formats with this ability, so the problem of attributing unexpected failures is mitigated to a certain degree. That being said, due to the extraordinary circumstances and challenges that are presented by this single Pokemon, Zoroark is banned from this competition.

H. Simulation Cost

The simulation cost is an important factor for agents with complex strategies or state search mechanisms. Simulation of a single turn using this framework takes roughly 5-40 milliseconds, depending on the complexity of the events that occur in the turn. Due to the high cost of simulation, it will

be difficult for agents to explore a large number of states within the allotted 20 seconds of decision time. This presents a practical challenge for agents that operate by simulating many turns, such as Monte Carlo Tree Search (MCTS). It is worth noting that damage prediction and usage of the reference API described in IV-A are much faster than full-scale simulation.

VI. Domain Knowledge

Pokemon battling contains a number of subsystems and domain-specific considerations that players make use of in decision making. It is possible for agents to also leverage this information to expedite decision making or improve strategy.

A. Typing

The type chart is perhaps one of the most fundamental aspects of the Pokemon combat system. All moves in the game are assigned a type, and all Pokemon have one or two types. When a move is used on a Pokemon, a multiplier is applied after the damage calculation step based on these types. If the moves type is super-effective against the targets type, the damage dealt is doubled, If the target is resistant, the damage is halved. Immunity nullifies the the attack altogether. For targets with multiple types, the multiplier is calculated for each type, and then multiplied together, potentially resulting in 4x or 0.25x multipliers. There is an additional multiplier of 1.5x that is applied if the users type matches the moves type, known as Same Type Attack Bonus (STAB). The bonuses and penalties associated with type are quite large, and so super-effective moves generally will do more damage than resisted moves. This domain knowledge allows for the estimation of damage and move value without requiring the use of simulation which, as discussed earlier, is expensive.

B. Archetypes

Introducing domain knowledge can greatly mitigate the cost of dealing with hidden information. For instance, there are several pairs of moves that are well known to synergize. If a grass type Pokemon is revealed to know the move Sunny Day, it is generally reasonable to assume they have access to the move Solarbeam, as it has an effect that directly benefits from Sunny Day. Additionally, many Pokemon fall under certain archetypes, and will be built to use moves that leverage their strongest qualities. For instance, it is generally safe to assume that a pokemon with exceedingly high physical attack and subpar special attack will almost exclusively use physical moves to deal damage. If one were to incorporate domain knowledge at a higher degree of specificity, it would be possible to predict moves based on what archetypes specific Pokemon fall into. For example, there is a set of Pokemon in competitive play that fall into an archetype known as Spinners. These Pokemon are generally valued for their access to Rapid Spin. It is generally safe to assume in competitive play that Spinners will have Rapid Spin in their movesets.

On a macroscopic level, domain knowledge can be useful in predicting what Pokemon an opponent has. It is important to note that the following usage of domain knowledge is only applicable in formats wherein players build their own teams. Manually built teams are often with a core strategy in mind. Being able to identify an opponents core strategy allows players to predict the enemy team with a certain degree of accuracy. For example, Trick Room is a move that reverses the order Pokemon attack each turn. If a player uses this move, it is generally safe to assume they are using a number of hard-hitting but very slow Pokemon.

C. Delayed Rewards

Several strategies in Pokemon involve the use of moves that provide benefits long after the current Pokemon have been defeated. For example, Spikes, Toxic Spikes, and Stealth Rock are moves known as entry hazards, which damage Pokemon whenever they switch into battle. When used properly, a Pokemon can take up to 75 percent of its maximum health in damage upon switching into battle. These moves are often non-damaging and offer offer no short term benefits, and so may be viewed as low value by players who value short term gains. However, there is general agreement among the competitive community that these moves are valuable assets. Long-range foresight is a key part of fully understanding the ramifications of moves in Pokemon, and knowledge about how moves will play out is essential in playing at a high level.

VII. Sample Controllers

Several sample agents were developed to demonstrate the development of players using the Showdown AI framework as well as introduce preliminary results regarding the comparative performance of several basic algorithms.

A. Breadth-First Search

Breadth-First search is a basic tree search algorithm. Given a root *Battle* object representing the current game state, this algorithm explores the outcomes of all possible choices, treating these resultant states as child nodes. BFS traverses these nodes in level order until it finds a state in which the current opponent Pokemon is fainted. As a non-adversarial algorithm, the agent selfishly assumes that the opponent uses ”forceskip” (as outlined in Section IV-B) as its choice each turn.

B. Minimax

Minimax is a tree search algorithm that deals with adversarial paradigms by assuming the opponent acts in their best interest. Each node in this tree represents the worst case scenario that would occur if the player had chosen a specific choice. The agent also uses an alpha-beta pruning strategy to ignore any node in which any of the agent's Pokemon faints. The tree itself is traversed using a greedy strategy, which terminates under the same conditions as BFS. Both the traversal order and worst-case evaluation are performed using the following evaluation function:

$$Eval = \frac{hp_{myPoke}}{maxhp_{myPoke}} - 3 * \frac{hp_{oppPoke}}{maxhp_{oppPoke}} - 0.3 * depth$$

	Random	BFS	Minimax	SLP	MLP	OTL	TS	PBFS
Random	N/A	0R 15B	0R 13B	3R 37B	4R 33B	0R 13B	0R 23B	0R 15B
BFS	**15R 75B**	N/A	2R 34B	**11R 66B**	**9R 60B**	1R 26B	**10R 65B**	3R 31B
Minimax	**15R 77B**	**9R 58B**	N/A	**14R 80B**	**10R 58B**	2R 30B	**10R 65B**	3R 31B
SLP	6R 53B	1R 24B	0R 10B	N/A	3R 30B	0R 7B	1R 21B	0R 20B
MLP	**10R 57B**	1R 30B	3R 32B	**8R 60B**	N/A	0R 8B	6R 45B	0R 25B
OTL	**14R 77B**	**10R 64B**	**9R 60B**	**15R 83B**	**13R 82B**	N/A	**12R 68B**	4R 43B
TS	**12R 67B**	1R 25B	3R 37B	**13R 69B**	5R 45B	0R 21B	N/A	3R 38B
PBFS	**15R 75B**	**10R 59B**	5R 48B	**12R 70B**	**10R 65B**	7R 47B	6R 52B	N/A

TABLE I

THE RESULTS OF PLAYING THE EXAMPLE AGENTS AGAINST EACH OTHER. EACH CELL REPRESENTS HOW MANY ROUNDS (R) OUT OF 15 AND BATTLES (B) OUT OF 90 THE AGENT IN THE ROW WON AGAINST THE AGENT IN THE COLUMN.

This is a fairly knowledge-free evaluation function that heavily favors dealing damage to the opposing Pokemon, while also rewarding survival. The depth penalty exists to promote exploration and discourage excessive turn depth.

C. Q-Learning

Q-Learning is a reinforcement learning algorithm designed to have an agent learn the potential values of moves. This is done by having the agent output the expected reward corresponding to an input *(State, Action)* pair, and updating the weights once actual reward values are found. Two agents were developed using Q-Learning: a single layer perceptron and multi layer perceptron. These agents were rewarded for defeating an opponent's Pokemon and punished for allowing one of its own pokemon to faint. Because decisions made tend to have long term consequences, weights are updated using the last three *(State, Action)* pairs rather than the most recent pair only. Additionally, in order to promote exploration, the agent employs an epislon-greedy selection policy, causing it to randomly override its decision with a probability of 0.1. The single layer perceptron was trained using the Delta Rule, while the multi player perceptron was trained using Delta Rule plus Backpropagation.

D. One Turn Lookahead

One Turn Lookahead is a heuristic based agent designed to encapsulate a greedy strategy that prioritizes damage output. This is a fairly popular strategy at lower level play, although it has been shown to be effective at that tier. The agent operates by estimating the damage dealt by all usable moves, including those usable by the agent's inactive but usble Pokemon. If the highest damaging move belongs to the active Pokemon, the agent will use that attack. If the most damaging move belongs to an inactive Pokemon, the agent will switch to that Pokemon.

E. Type Selector

Type Selector is a variation upon the One Turn Lookahead agent that utilizes a short series of if-else statements in its decision making. The agent begins by iterating through all moves usable by the current Pokemon, and using the frameworks *getDamage* function to estimate damage. If this damage is greater than the opponents current health, then this signifies a clear path to victory, and as such the agent will return this move. If there is no move in the Pokemons repertoire, it evaluates the favorability of the Pokemons typing. If the type matchup between the current active Pokemon and the opposing Pokemon is determined to be acceptable, the agent will return the most damaging attack it found in the initial search. If the type matchup is found to be undesirable, the agent will iterate through the Pokemon it can switch to. It will then switch to the Pokemon with the most favorable type matchup against the current opponent.

F. Pruned BFS

This agent is designed to demonstrate a simple way to utilize domain knowledge as a cost-cutting measure. This algorithm does so by making modifications to the Breadth First Search agent. First, the algorithm does not simulate any actions that involve using a damaging move with a resisted type, nor does it simulate any actions that involve switching to a Pokemon with a subpar type matchup. Additionally, rather than selfishly assuming the opponent skips their turn in each simulation, the agent assumes its opponent is a One Turn Lookahead agent and simulates accordingly.

VIII. RESULTS

Each agent played 15 rounds of six games each against every other agent in a similar fashion to the final round outlined in Section III. The first number in each cell represents the number of rounds the row agent win against the column agent out of 15. The second number represents the number of battle the row agent won against the column agent out of 90. Ties are not counted, both for rounds and battles, and so the numbers may not necessarily add up to 15 and 90 respectively. A one-tailed binomial test was performed upon the number of games won for each matchup. If the resultant p value is less than 0.05, the value is bolded in the table, indicating statistically significance superiority.

From this table, it can be seen that Minimax, One Turn Lookahead, and Pruned BFS significantly outperform the other agents in this set. While Pruned BFS appears to slightly outperform One Turn Lookahead, its weaker performance against other agents appears to perhaps suggest a strategic rock-paper-scissors paradigm, where PBFS's strategy happens to counter that of OTL, but performs worse against other strategies. This likely stems from the fact that PBFS uses an OTL agent as its opponent model, meaning the agent in a way is designed to play against OTL.

A possible explanation for the weakness of BFS, Minimax, and Pruned BFS lies within the competition's 20 second time

limit. Due to the cost of simulation as well as the large search space, the tree search algorithms time out fairly often. This is especially true if the player's Pokemon is afflicted with a status that causes it to skip turns, as that Pokemon will not be able to make any progress toward its goal until the status is cleared, wasting entire layers in the tree search.

One Turn Lookahead has demonstrated strong performance among the agents. It is possible that among the toy metagame established by these agents, an aggressive damage optimization strategy is strongest. However, human play has established that this is an easily countered strategy in upper level play, and so there is a possibility that the performance of this agent will drop significantly as more complex strategies are implemented.

One particular point of interest is the comparison of the Minimax and Breadth First Search agents. These two agents represent a direct trade-off between higher search depth and more accurate prediction.

IX. Conclusion & Future work

The initial results presented here provide some basis for predictions into how agents may perform in this and future iterations of this competition. In the long term, predetermined heuristic-based agents will likely suffer. The complexity of a heuristic agent is directly proportional to the complexity of the strategy it employs. Implementing complex strategies with priority lists and decisions trees by hand is likely to reach diseconomies of scale at some point. Conversely, considering the low computation cost and relatively high performance of the One Turn Lookahead agent, there is possibly potential for the automated training or evolution of strategic agents.

There is evidence to imply that simulation accuracy is more valuable than search depth in this domain, and as such the simulation strategy will be very important here. However, as Minimax has demonstrated, thorough simulation is costly and greatly limits the number of explored nodes. Methods by which domain knowledge is used to create close approximations at mch lower cost will likely be useful moving forward. As Pruned BFS has demonstrated, there are cost effective ways to reduce the number of simulations in a tree search, and there are undoubtedly other angles from which domain knowledge can enhance tree search.

There is some potential for deep learning algorithms as well. The Single Layer Perceptron's dominance over the random agent implies there is some pattern that can be learned. For the time being, deep reinforcement learning seems to be an effective direction for these algorithms. As Pokemon is a new domain for AI, there is little training data to work with, making offline learning problematic. One particularly interesting avenue for machine learning may be the training of state evaluators or in a sense neural forward models. By using models to predict output states or evaluations, once could potentially leverage the thoroughness of a tree search algorithm while drastically cutting down on simulation costs.

A. Future Competitions

While randomly generated teams have the benefit of reducing the engineering workload for AI developers, it does have the side effect of impacting the game's strategic landscape[18]. While the randomized format tests an agent's ability to deal well with a hand they are dealt, giving an agent the ability to build their own team tests the ability to formulate the optimal hand. An interesting direction for the Showdown AI Competition is the introduction of a team building track. To the author's knowledge, there are currently no game AI competitions with a substantial team or deck building component. This paradigm brings with it a proactive strategic element that takes effect before a game even starts, and introduces the problem of compatibility. Which Pokemon a player has is significant to what strategies they can execute, and so creating a team that is compatible with the battling agent and vice versa will be a key challenge in this domain.

References

[1] J. Togelius, N. Shaker, S. Karakovskiy, and G. N. Yannakakis, "The mario ai championship 2009-2012," *AI Magazine*, vol. 34, no. 3, pp. 89–92, 2013.

[2] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[4] S. Matsumoto, N. Hirosue, K. Itonaga, K. Yokoo, and H. Futahashi, "Evaluation of simulation strategy on single-player monte-carlo tree search and its discussion for a practical scheduling problem," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 3, 2010, pp. 2086–2091.

[5] J. C. Rosser, P. J. Lynch, L. Cuddihy, D. A. Gentile, J. Klonsky, and R. Merrell, "The impact of video games on training surgeons in the 21st century," *Archives of surgery*, vol. 142, no. 2, pp. 181–186, 2007.

[6] M. Treanor, A. Zook, M. P. Eladhari, J. Togelius, G. Smith, M. Cook, T. Thompson, B. Magerko, J. Levine, and A. Smith, "Ai-based game design patterns," 2015.

[7] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, pp. 317–335, 2015.

[8] J. Togelius, "How to run a successful game-based ai competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 1, pp. 95–100, 2016.

[9] P. Rohlfshagen and S. M. Lucas, "Ms pac-man versus ghost team cec 2011 competition," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, 2011, pp. 70–77.

[10] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lonneker, L. Cardamone, D. Perez, Y. Sáez *et al.*, "The 2009 simulated car racing championship," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 131–147, 2010.

[11] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[12] Game Freak, *Pokemon Series*. Nintendo, 1996-2017.

[13] ——, *Pokemon Red and Blue*. Nintendo, 1996.

[14] Overused. [Online]. Available: http://www.smogon.com/dex/xy/formats/ou

[15] Gen vi tiers. [Online]. Available: http://www.smogon.com/xyhub/tiers

[16] S. Lee. Showdown ai client. [Online]. Available: https://github.com/scotchkorean27/showdownaiclient

[17] G. Luo. Pokemon showdown. [Online]. Available: https://github.com/Zarel/Pokemon-Showdown

[18] Oglemi. Randbats - how to play from an expert.