Combining Reinforcement Learning and Monte Carlo Search Algorithms for Pokémon Battling

Final Report

Christopher Davis

Abstract

Pokémon is a very popular media franchise that targets a younger audience. However, a great deal of depth exists within its battling system. Many strategies exist and a good player will alter their strategy over the course of a battle. A Pokémon battle is a multi- agent, partially observable domain where agents must reason simultaneously, factoring in each other's possible and likely actions. This project aims to develop an AI agent that can dynamically combine reinforcement learning and Monte Carlo search algorithms that can compete against skilled human players.

Keywords

Artificial Intelligence; Reinforcement Learning; Q Learning; Monte Carlo Search; Pokémon

Contents

Pokémon Game Explanation

Pokémon is a well-known series of video games developed by the company Game Freak. Within the game, the player takes on the role of Pokémon trainer (hereafter referred to as player or trainer). The top level of gameplay consists of traveling the game world where the player can encounter creatures called Pokémon. Players can obtain these Pokémon and can have up to 6 in their team at any given time. Throughout the game, the player will encounter other trainers and these encounters lead to a Pokémon battle. Pokémon battles are also accessible both in the official video games as well as through unofficial simulators in a multiplayer format, allowing two human players to battle against each other. This can be done casually or as part of unofficial and official competitive tournaments. Battles against human players are much more challenging the battles available in the official games, requiring the effective use of more game mechanics and strategies.

Pokémon battles as a two-player competitive game form a sub level of gameplay which is the focus in this project.

Pokémon Creature Description

A Pokémon is defined through several elements. Some of these elements are shared by all Pokémon of the same species whilst other elements vary between different instances of the same species. The following paragraphs will go through these elements.

Type

Each Pokémon species has up to two types. Any given pair of types will have 2 multipliers representing the effectiveness of each type on the other – known as type effectiveness. For example, the electric type has a 2 times modifier targeting the water type whilst the water type has a 1 times modifier against the electric type. The possible multipliers are 1 (neutrally effective), 2 (super effective), 0.5 (not very effective) or 0 (not effective). In the case of Pokémon with two types, the overall multiplier is determined by multiplying the modifiers of the individual types. Seeking advantageous type match ups is often a good strategy in battles. [1]

Item

A Pokémon can hold an item in battle. Items have a broad range of effects that occur passively, rather than being chosen to take effect by a player. [2]

Ability

Each Pokémon species has access to between 1 and 3 abilities with an individual Pokémon of that species having one of these abilities. Abilities function similarly to items in that their effects are not manually triggered. [3]

Moves

Each Pokémon has up to four moves that the player chooses from a larger pool. Moves are actions that a Pokémon can undertake in a battle. Moves belong to one of three categories. Physical and Special moves inflict damage directly to the opponent. The difference between physical and special moves are the stats that are used to determine the damage inflicted. The third category is status moves. Status moves don't deal direct damage and instead have a wide range of effects including but not limited to increasing the user's stats, decreasing the opponents' stats, restoring the user's health, setting up entry hazards that deal damage to opposing Pokémon that switch in or applying status conditions. The listed effects are amongst the most common.

All moves have a type, a base power value and an accuracy. The base power of a move is number between 0 (status moves have a base power of 0 as they do not inflict direct damage) and 250. The base power is part of the formula that calculates the damage a move inflicts. When the type of a move matches one of the types of the Pokémon using it, the base power is multiplied by 1.5 which is known as the same type attack bonus. Base power is also multiplied by the type effectiveness multiplier of the moves type against the opposing Pokémon's type.

Moves also have a limited number of uses, known as Power Points (PP). Each use of a move uses up one PP and when the PP for a move reaches 0, the move can no longer be used. The maximum PP of moves ranges from 8 to 64, with more powerful moves having fewer PP. In a competitive setting, most moves will have either 8 or 16 PP. For most battles, running out of PP is not a concern. [4]

Stats

Pokémon have 6 stats. A species has integer base stats which define the range of possible integer values the actual stat can take for a Pokémon of that species. **HP** is the amount of damage the Pokémon can take before fainting. **Attack** and **Special Attack** affect how much damage the Pokémon inflicts with its physical and special moves respectively. **Defence** and **Special Defence** affect how much damage the Pokémon receives from opposing physical and special moves respectively. and **Speed** affects the order in which moves are carried out. [5]

With the exception of HP, stats can be boosted in battle. Stat boosts are typically referred to as plus or minus a number from 1 to 6. Plus 6 is the maximum level of stat and minus 6 is the minimum. Until a change is made, a stat is considered to be at plus 0. Some status moves can increase or decrease stat levels. For example, a move may increase a stat by plus 2 so a Pokémon at plus 0 would go to plus 2, a Pokémon at minus 1 would go to plus 1 and a Pokémon at plus 5 would go to plus 6 which is the maximum.

Effort Values (EVs)

Effort values are one of the factors that contribute to the range of possible stat values a Pokémon has. For each stat, a Pokémon will have an integer number of EVs from 0 to 255 assigned to that stat. Every 4 EVs assigned to a stat increases the stat by 1 aside from some uncommon instances where the stat increases by 2. A Pokémon can have no more than 510 EVs assigned across all of its stats. Each player chooses the allocation of the 510 EVs for each of their Pokémon and whilst some Pokémon will have an allocation of EVs that a majority of players use, two Pokémon of the same species can perform differently due to differing EV allocations.

Individual Values (IVs)

For each of its 6 stats, a Pokémon will have an individual value for that stat as an integer between 0 and 31 inclusive. The higher the IV, the greater the stat. Typically, most Pokémon used in battles will have 6 IVs all at 31 and any IVs that aren't 31 will be 0, typically due to the Pokémon's performance not being reliant on that stat. IVs typically are less variable than EVs across different teams.

Nature

A Pokémon will have one of 36 natures. Each nature increases one stat by 10% and reduces another by 10%. 6 of the 36 natures increase and decrease the same stat, making them neutral overall. These natures are not used as almost all Pokémon will only rely on either the attack stat or the special attack stat, leaving the other attacking stat unneeded and safe to be reduced by a nature. For many Pokémon, the choice of nature is between increasing speed or increasing an attack stat and this choice affects how a Pokémon should be used and battled against.

Status Condition

At any given point, a Pokémon can be afflicted with one of 6 primary status conditions. Other status conditions exist but are rarer, either limited in use to certain Pokémon or not used as part of a strategy for battle. Status conditions can either be caused due to status moves, abilities or as a secondary, chance-based effect from a physical or special move. Status conditions persist when a Pokémon switches out to a different Pokémon and with two exceptions, status conditions can only be removed by specific status moves, abilities and items. The primary status conditions are as follows:

Poison and **Badly Poisoned**: Poison causes a Pokémon to lose 12.5% of its HP at the end of every turn whilst Bad Poison causes a Pokémon to lose an increasing amount of HP at the end of every turn, starting at 6.25% and increasing by 6.25% each turn. This increase resets if a Pokémon switches out from the battle and starts again when the Pokémon is next in battle.

Sleep: Pokémon that are asleep cannot move unless they use the moves Snore or Sleep Talk, which are uncommon. A Pokémon will wake up after a random number of turns between 1 and 3. A Pokémon can use a move on the turn it wakes up.

Burn: A Pokémon that is burned will lose 6.25% of its HP at the end of the turn, as well as the power of physical moves being halved.

Freeze: A Pokémon that is frozen cannot use any moves until they thaw. A Pokémon has a 20% chance each turn to thaw. Certain moves, especially fire moves can thaw a Pokémon that is either targeted or using the move itself.

Paralysis: A Pokémon that is paralyzed has its speed stat halved and has a 25% chance to fail its move each turn.

Pokémon Battle Format

There are multiple types of battle. In this project, single battles are used. A single battle has two opposing trainers. Each trainer has one Pokémon as their 'active' Pokémon whilst the rest of the trainer's team of six is in reserve. [6]

The battle is turn based and each turn begins with both trainers choosing their actions simultaneously. The set of actions consists first: of exchanging their active Pokémon for each Pokémon in their team (called switching) and second: of the active Pokémon carrying out one of the up to four 'moves' that the Pokémon knows. Once both players have chosen their actions, the game carries the actions out, showing the player the outcome. Typically, the Pokémon with the higher speed stat carries out their action first but switching will occur before any move occurs and some moves will be carried out first regardless of the speed of the respective Pokémon.

The objective of both players is to render all the opponent's Pokémon as fainted. A Pokémon enters the fainted state when its current HP reaches 0. Fainted Pokémon cannot use moves and cannot be switched in. The first player to cause all the opponents Pokémon to faint is the victor. Should both players final Pokémon faint on the same turn, a single victor will be determined through the specifics of how each Pokémon fainted. Battles cannot be tied, there will always be a winner and a loser.

More specifically this project uses the Smogon Gen 8 OU format for single battles. [7] This format follows the same structure of a single battle as previously described but sets rules regarding the Pokémon, moves and abilities which can be used, as well as some additional rules. Some rules reduce the impact of randomness of battles. An additional feature of these battles is a turn timer. By

default, there is no turn timer and so each player has as much time as they wish to select their move. However, either player can unilaterally enable the timer, giving each player 150 seconds to select their action. If the timer runs out, the player that didn't select an action loses via automatic forfeit. A player who nearly reaches the end of the timer will have a reduced timer in future turns, potentially leaving a player with only 30 seconds to select an action.

Dimensions of Complexity

Dimensions of complexity are a framework that can be used to define aspects of an agent developed for a particular problem space. Considering Pokémon battles, the dimensions of complexity are:

Indefinite Stage: Pokémon battles have no fixed length. It is possible for a regular single battle to last infinitely but the Smogon ruleset [7] bans players from creating infinite battles, making this format indefinite stage. Therefore, an agent cannot fully rely on a fixed length plan to win battles.

Partially Observable: At the beginning of a battle, each player can observe the species of all Pokémon on the opponent's team, but the player cannot see anything else. Over the course of the battle, a player can learn more about the opponent's team by observing the actions the opponent takes and the effects those actions have. Most battles however will finish without players learning everything about their opponents' team.

Stochastic: A Pokémon battle moves from state to state by both a player and an opponent's action. Therefore, an agent cannot determine with certainty the state that will arise from their own action as the state will also be affected by the opponent's action. Furthermore, there are many random elements in Pokémon battles. Many moves have a chance to miss the opponent: having no effect. Many moves have additional effects that can have a chance to occur or not. A deterministic formula calculates the damage done by a direct attack, but this damage is multiplied by a random number between 0.85 and 1. Players should observe the outcome of actions and then adjust as necessary their plans.

Multiple Agent: Pokémon battles are 2 player games, with agents working against each other. Predicting and planning around an opponent's actions is an important aspect of battling.

Online Reasoning: When creating a team, a player will think about strategies to fit the team and may create plans for how to battle with the team. However, online reasoning is needed due to an opponent's team not being known ahead of the battle as well as the unpredictable nature of Pokémon battling.

Project Motivation

Many games have seen artificial intelligence used to play them. Two notable examples are Alpha Go [8] and Alpha Zero [9]. Alpha Go is a neural network system, trained to play the game of Go. Go is a fully observable, deterministic game but has a sufficiently large search space such as to make exhaustive searches impractical. Alpha Go, through a combination of Reinforcement Learning and Monte Carlo simulations amongst other elements has been able to win games of Go at the highest level, defeating champion players [10].

Alpha Zero also uses reinforcement learning and Monte Carlo tree search for additional games such as chess; another fully observable, deterministic game. Again, a high level of play was reached, reaffirming the effectiveness of these approaches in these sorts of games.

The success of these approaches in chess and go does not necessarily mean success in non-deterministic and partially observable games, such as Pokémon battling.

Project Background

Previous research exists in the field of Pokémon battling artificial intelligence. One such paper [11] compared a range of artificial intelligence algorithms across 3 scenarios of Pokémon battle of increasing complexity. The first scenario had both players with a single identical Pokémon with 2 moves, one that would deal damage and another that would decrease the defence of the opposing Pokémon and thus increase the damage of the first move on subsequent turns. The second scenario had both players again with one Pokémon, but the players had different Pokémon with slight differences in the available moves. The third scenario had both players with the same team of 6 Pokémon.

The artificial intelligence algorithms used were a purely random agent, an agent with a pre-set, hard coded sequence of actions to take, a linear combination of features agent that was trained to determine weights and a Monte Carlo search agent.

The simplest scenario has an optimal strategy, allowing the hardcoded agent to perform best, however the Monte Carlo search agent performed well across the varying complexities, unlike the other agents that performed worse in the more complex scenarios. The Monte Carlo agent was able to reach a near 50%-win rate against human players, albeit more experienced players won more often against the agent.

Another paper [12] uses a reinforcement learning agent, specifically a Q learning algorithm. The agent was tested against a Knowledge Based System in two scenarios: one where both agents had the same team of 6 and the other where the agents had separate teams. In the first scenario, the Q learning agent had a near 50%-win rate compared to a 29%-win rate in the second scenario

Project Objective

From previous research, Monte Carlo search and reinforcement learning both demonstrated potential in the domain of Pokémon Battling. These techniques had been used as part of larger systems for Alpha Go [8] and Alpha Zero [9] for other games which raised the question if combining the techniques could provide greater performance than one alone for Pokémon Battling. A range of agents were developed using Q learning and Monte Carlo search algorithms combined and in isolation alongside other simpler agents to test against. Furthermore, these agents are tested in a format used by human players for serious competitions, rather than a simplified scenario to give a more realistic view of their abilities.

Ethical Concerns

Agents were tested against human players, but no personal information was viewed or stored. The only information observed from the human players were pseudonymous usernames which were not stored. The only information from these battles that was stored related to the battle states. Therefore, there were no ethical concerns.

Agent Design

Max Damage and Max Damage Plan Agent

These agents were designed to be simple agents for the reinforcement learning and Monte Carlo agents to be evaluated against. The Max Damage agent will never switch and will pick the available move with the highest base power. When a choice is required where there are no available moves to be chosen, such as selecting a Pokémon to replace the active Pokémon if it faints, a random choice is made.

The Max Damage Plan agent is a more complex version of this agent. First, the agent determines the type effectiveness of the opposing active Pokémon's first typing against its active Pokémon. If this modifier is greater than 1, it indicates the opponent has an advantage and so the agent switches to the non-fainted Pokémon that has the lowest type effectiveness modifier against the opponents active Pokémon in pursuit of an advantage match up.

If the condition for switching is not met, the agent will pick the move that has the highest base power. However, unlike the Max Damage agent, the Max Damage Plan agent will apply relevant multipliers for same type attack bonus and type effectiveness.

Neither agents will use status moves unless an error or unexpected situation arises that causes a random move to be chosen.

Q Learning Agent

Q Learning was chosen as for the reinforcement learning algorithm due to its relative simplicity and ease of implementation. The possible state space too large to construct a full table for state action pairs so a more dynamic approach was used. Each turn, the agent observes the current battle state, embedding this observation into a JSON object. A knowledge base in the form of a JSON file containing a list of JSON observation objects is checked to see if the observation is in the list. If the observation is in the list, then the current battle state has already been observed. If the observation isn't in the knowledge base, it is added with the possible actions for the state. The JSON file knowledge base forms the Q table. The knowledge base is formatted to ensure readability via full word elements and values rather than smaller encodings to aid in development. This was considered to be worth the consequently larger file size.

The gamma value for exploration was set to 20%. This encourages exploitation more which was done to prioritise refining the Q value for a given action due to the rarity of encountering the same state. Therefore, a state would be more likely to have an accurate Q value for one action, rather than very inaccurate values for all actions.

Exploration results in a random action being taken, combining the possible moves and the possible switches. Initially, there is a greater chance to switch than to attack due to the first turn having 5 possible switches and 4 possible moves, but this changes later in the battle as the number of moves available will in most cases remain at 4 whilst the number of possible switches can fall to 0 as the agent's Pokémon faint. Exploiting results in the action with the highest utility value being chosen.

A class variable whether the current turn is the first turn of the battle. This is used to indicate that the agent should update the Q value of the previous action taken using the utility of the current state which will be the effect of the previous action in the previous state. The previous action and observation are both stored as class values and are tracked, and action utility updated regardless of whether the agent explores or exploits.

State Observations

The state observation consists of the following factors.

Active Pokémon: The species of the active Pokémon for both the agent and the opponent is observed, as well as the current HP in the form of a decimal proportion of maximum HP rounded to 1 significant figure. The HP rounding goes for all Pokémon in the observation and is done to reduce state space otherwise there would be 101 possible percentage HP values for each Pokémon. If rounding the HP value would round down to 0, HP is instead set to 0.01 as a HP of zero indicates a Pokémon being fainted. The difference between 54% and 53% HP is not significant and is unlikely to be a useful distinction whereas the difference between 4% HP and 0% is significant and so should be differentiated Any stat boosts and status conditions on the active Pokémon are also observed.

These observations are important due to the active Pokémon determining which 4 moves either player can make.

Inactive Pokémon: For Pokémon in both the agent and opponents' team, the species, rounded HP and status conditions are observed. Stat boosts are lost when a Pokémon switches out, so they don't apply to non-active Pokémon. For the opposing team, the held item is also noted, with the item being listed as 'unknown' if the item has not yet been revealed. The way an agent may use a Pokémon can vary with the status of the two teams. A Pokémon may be more or less valuable and thus played more or less safely and conservatively depending on how it performs against the opponent's team.

Entry Hazards: There are status moves that set up a condition that apply an effect on an opponent's Pokémon when it switches in. The agent observes two of the most common entry hazards: Stealth Rock and Spikes. Stealth Rock is either in effect or not whilst Spikes can be at one of three levels of effectiveness or not in effect.

Utility

The elements of the observation are used to calculate the utility of a particular state. The below table figure 1 shows the feature and the contribution to state utility of that feature.

Feature	Utility
Pokémon HP	(1.0 – current rounded HP) for each Pokémon
Pokémon Status Conditions	0.2 for each Pokémon with a status condition
Pokémon Stat Boosts	0.05 for each level of boost for each stat
Stealth Rock	0.2 if active
Spikes	0.1 per layer of active spikes

Figure 1: Feature Utility Table

HP and status conditions are subtracted when calculated for the agent's team and added for the opponent's team, rewarding inflicting damage and status conditions to the opponent and punishing receiving damage and status conditions from the opponent.

Stat Boosts, Stealth Rock and Spikes are added for the agent's team and subtracting for the opponent's team, rewarding boosting the agent's Pokémon's stats, reducing the opposing Pokémon's stats and setting and keeping up stealth rocks and spikes while punishing having the agent's Pokémon's stats reduced, letting the opponent boost their own stats and letting the opponent maintain their own stealth rock and spikes.

The utility of an action in a state is initialized to 1 when the state is first observed. From then on, the utility is updated by adding the difference in utility of the state and the resulting state from the action.

Q Learning Lite Agent

The state space used by the Q Learning agent was infeasibly large for any Q values to converge and so a version was developed to reduce the complexity of observations. The Q Learning Lite agent observes the rounded HP of both the agent and the opponents active Pokémon, the stat boosts of the agent's active Pokémon and whether or not the agent's non active Pokémon are fainted or not. This drastically reduces the state space of the agent, increasing the chance that two identical states are encountered allowing for Q values to be updated more frequently.

The utility function was modified accordingly. The same values are used but only the features in the new observations affect the utility.

The explore chance was set to the same 20% as the previous Q Learning agent for the same reasons outlined above. When a Q value is updated, the formula adds the change in utility from original state to new state as well as 0.8 times the difference in the max utility of the new state and the utility of the action in the new state. This ensures that the utility of taking an action is affected both by the utility of the resultant state and the utility of the best possible action in the resultant state. Therefore, increasing utility is punished if the resultant state leads to worse states, encouraging the agent to make progress toward victory, rather than performing actions focusing solely on improving short term utility.

The general structure of the Q Learning Lite agent other than the aspects described above is the same as the original Q Learning agent.

Monte Carlo Agent

The Monte Carlo agent simulates possible outcomes of choosing actions from both the agent itself and the opponent. These simulated actions are used to create a plan of actions to take in the form of a queue of actions and corresponding states within which the actions are taken.

Node Format

The Monte Carlo stores states and actions as nodes in a search tree. Each node has the following attributes.

State: Stores the battle state the node represents.

Parent: Points to the node that represents the previous state

Action: The action taken in the parent node state that results in the state of this node

Children: The list of nodes whose states are a result of using an action in this state.

End State: A Boolean to mark if the node is an end state and thus should remain a leaf.

Results: Stores the number of wins and losses that result from the node's children.

Untried Actions: Stores the list of actions possible in the state that have not yet been used to simulate and produce children.

Search Tree Creation

At the beginning of a battle, an empty plan is initialized. On the beginning of a turn, if the plan is empty, the Monte Carlo agent initializes a tree with the root node being built from the current state. The agent carries out a breadth first search of the tree, generating child nodes by simulating all the possible pairs of actions to move from state to state from itself and the opponent. Initial plans were to use the Max Damage Plan agent to choose the likely actions for the opponent for simulation but this idea was abandoned due to difficulties in implementation as well as the concern that the Max Damage Plan player could not be expected to perform as well as a human player which would affect the performance of the Monte Carlo agent. Breadth first search is used as the search tree has infinite depth: if both trainers always switch, without ever using a move, the battle will last forever as no damage would ever be inflicted and so no Pokémon would ever faint. The rules of the format prohibit this, and it requires intentional effort from both players, so it is an unnecessary consideration for real battles, but the agent simulations consider all possible actions, including both players switching so to avoid an infinite descent like this, breadth first search is used.

Due to the potential for a turn timer, the tree construction runs until either 5000 actions have been simulated or 50 end states have been encountered, whichever comes first. An end state is a state wherein either all of the agent's Pokémon are fainted or all of the opponent's Pokémon are fainted, resulting in a win in the former case or a loss in the latter case. Upon reaching an end state, the nodes on the path to the end state have their results attributed incremented either with an additional win or loss depending on which the end state was.

Due to Pokémon being a partially observable game, the Monte Carlo agent isn't able to perfectly simulate the possible actions of the opponent as the agent doesn't have access to the move choices of the opponent as well as information regarding actual stat values, items and abilities which can all affect the damage of moves. The Monte Carlo agent uses a file than contains all Pokémon that had a usage rate greater than 10% at time of file creation along with the four most used moves. The usage information was sourced from the website Pikalytics [13].

Plan Creation

Once the tree creation has finished, the agent creates a plan by traversing through the tree. At each node, it chooses the child node that has the greatest difference between wins and losses. This difference is not the absolute difference, so a node that leads to 6 wins and 3 losses will be chosen over a node with 6 loses and 3 wins. The plan contains each state and the action the agent should take in that state.

Choosing Action

When prompted to choose an action, the agent checks the plan. If the agent is in the first state of the plan, it will carry out the first action of the plan. If the plan is empty, or the state in the plan is not the same as the current state of the battle, the agent will regenerate the search tree and plan. This measure is done as the opponent will not necessarily play in a predictable optimal way and the plan with the best chance of success found by the agent may not be applicable if the opponent chooses different actions.

Hybrid Agents

The hybrid agents all use the actions selection methods from the Q Lite agent and the Monte Carlo agent. Initially they all use the Q Learning agent's selection method choosing the action with the greatest utility. Once some condition is met, the agent switches to using the Monte Carlo agent's method, creating a search tree and plan. This order was chosen as a Q learning agent should perform better when it is trained across multiple instances of choosing actions in a given state. The

beginning of a battle will have a smaller number of possible states which should result in states being revisited more often. The latter stages of the battle will a greater number of possible states when considering the entire set of possible states in the battle but given a root state there will be fewer possible children states as the battle is closer to being over with fewer actions needed to reach an end state so a lower average depth should occur. This would benefit the Monte Carlo approach as the condition of 50 end states should become easier to reach before 5000 simulations, meaning the plan should give better expectations of win loss difference for actions.

The Hybrid agents all draw from the same knowledge base as the Q Learning Lite Agent, meaning that training was not done separately for the hybrid agents.

Three hybrid agents were implemented with differing conditions for the switch to allow for comparisons within the hybrid approach. All of the switch conditions were chosen however to occur at a potential consideration for a halfway point of the battle.

Turn Switch

The first hybrid agent will switch once a certain number of turns was chosen. The length of battles can vary drastically depending on the skill of players and the types of teams being used. Some teams are slower paced, relying on status moves and indirect damage to stall and slowly defeat the opponent whilst other teams focus on direct attacks and finishing battles quickly. A value of 10 was chosen as the agents wouldn't be using defensive teams due to the higher difficulty and less immediately tangible effects of moves so 10 would be a reasonable estimate for midway through the battle.

The turn number was tracked through a simple class variable incremented each turn.

HP Percentage Switch

The second hybrid agent switches once a total of 300% damage is taken by the opponent's team. As all serious opponents will have six Pokémon on their team, 300% damage is effectively the midway point of the battle from the perspective of the agent's victory condition which would be inflicting 600% damage.

Each turn if the agent has not switched, the HP of the opponent's team is observed, and it is calculated if the opponent has lost at least 300% total health. If this condition is met, the agent switches to Monte Carlo.

Information Switch

The third hybrid agent switches based on information gained. Once all 6 Pokémon from the opponent have been used, the agent switches.

Common Design Elements

All agents extended the same player class. This results in the choose move and team preview methods being implemented. The choose move method is called every turn and returns an action. Each agent implements this differently. The team preview method is common to all agents.

The team preview method is called at the beginning of the battle. All players must choose a first Pokémon at the beginning of the battle which is returned by the team preview method. This is done by calculating the Pokémon with the greatest average type effectiveness against the opponent's team. For each pair of Pokémon, a and b, the maximum type effectiveness of a's types against b and the maximum type effectiveness of b's types against a is calculated. The overall type effectiveness is

given as the difference in these values. This ensures that, the agents first Pokémon has the greatest chance to have an effective typing against whatever Pokémon the opponent chooses first.

The team preview method is also used by some agents to initialise class variables for the battle.

An additional feature that the agents share is that when an action is selected, it is validated to ensure it is one of the valid actions that the agent can take. If the action is not valid, the agent can either try a different action or a default or random action can be taken instead. Submitting an invalid action to the server doesn't return a response that the agents receive so, if the battle timer is active, the agent will lose via time out. Therefore, providing any action is better as it can be no worse than not supplying an action at all.

Development

All development was done using the Python language due to prior familiarity and the Poke-env module [14]. This module provides classes and methods to represent elements such as Pokémon, moves and battles.

The first agent to be developed was a purely random agent, followed by the Max Damage and Max Damage Plan agents. Testing methods were developed to allow agents to battle against each other and against a human player for testing purposes. From the initial simpler agents, the other agents could be implemented. Each agent was implemented as its own class, inheriting from the Player class as defined in the Poke-env module. Methods were made modular where possible to speed development by reducing repeated code. Commonly used methods such as embed_battle, which converts the Poke-env battle object into a dict with the desired information, as well as the shared team preview method were placed into a separate utility file from the agent class files to aid in the organisation of the code.

The Q Learning and Monte Carlo choose move methods were also modularised to allow the hybrid agents to simply call those methods, meaning that the hybrid agents only needed to implement the logic to switch between choose move methods.

Throughout the project, GitHub has been used for version control and as a back up for all code written.

Project Management

To aid in the completion of the project, the objectives and associated tasks were set out in the beginning, and a timeline in the form of a Gantt chart was created, allocating time periods for working on and completing each task.

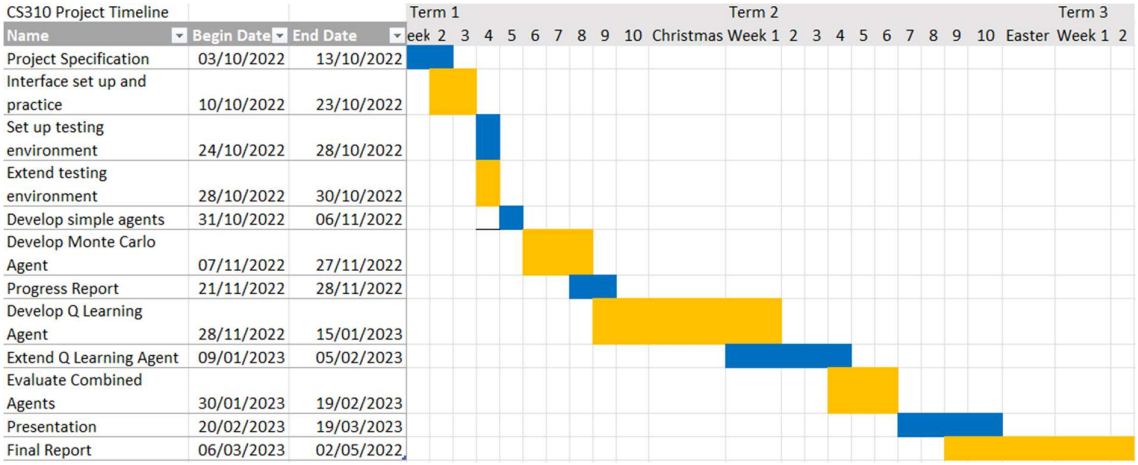

Figure 2: Initial Project Gantt Chart

The above figure 2 shows the Gantt chart created at the beginning of the project. Tasks were allocated with simpler tasks coming first to allow for familiarity to be gained with the Poke-env module and for testing of fundamental aspects to be done with simple agents.

Once the simpler tasks were done, more complex tasks were scheduled, such as implementing the Q Learning agent. An incremental approach to development was employed in regard to the agents, with simpler versions being developed and tested before then being extended. This provided

insurance that if tasks took significantly longer than expected, simpler versions of agents would still be usable for evaluation.

Name	Begin Date	End Date
Progress Report Draft	14/11/2022	20/11/2022
Progress Report Completion	21/11/2022	27/11/2022
Implement Q learning embed battle function	21/11/2022	27/11/2022
Implement Q learning utility prediction function	28/10/2022	30/10/2022
Implement Q learning update utility function	31/10/2022	06/11/2022
Implement complete Q learning agent	07/11/2022	27/11/2022
Implment Monte Carlo agent	21/11/2022	28/11/2022
Implment hybrid agents	28/11/2022	15/01/2023
Train learning agents	09/01/2023	05/02/2023
Cross evaluate agents	30/01/2023	19/02/2023
Evaluate agents against humans		
Presentation	20/02/2023	19/03/2023
Final Report	06/03/2023	02/05/2022

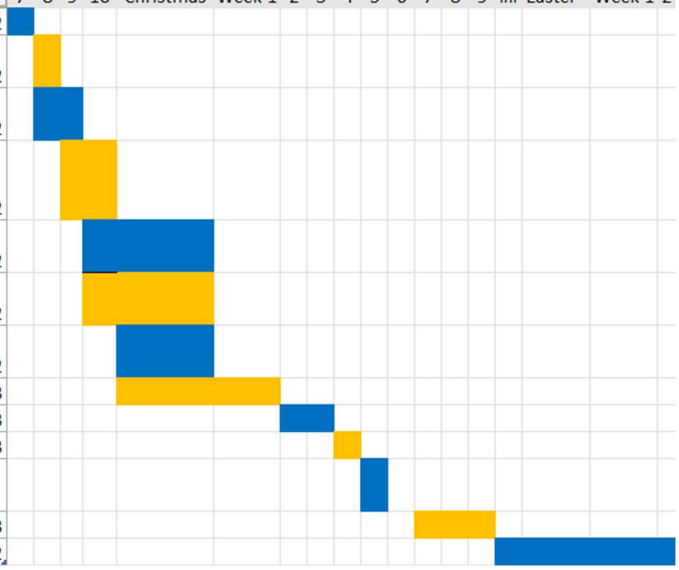

Figure 3: Updated Project Gantt Chart

The above figure 3 shows an updated Gantt chart created 7 weeks into the project. Progress had been made according to plan, but a change was made to prioritise the Q Learning agent due to its greater complexity. Developing a prototype version earlier in the course of the project would better inform the timeline going forward, allowing for a more accurate timeline.

This timeline stayed in place throughout the rest of the project. Some aspects such as the hybrid agents took longer than expected due to unforeseen complications regarding the modular methods. However, agent training and evaluation began earlier than scheduled, due to being carried out whilst the hybrid agents were still under development. This ensured that the project deadlines were still met.

Regular meetings with the Project Supervisor were maintained throughout the project. These meetings allowed for discussion over progress made and what next steps needed to be taken, helping maintain focus and direction.

Battle Performance Testing Process

All battle simulation was done using Pokémon Showdown [15]. Pokémon Showdown is a web-based Pokémon battle simulator, commonly used by players for competitive battling as opposed to using the official games due to various convenience factors. Poke-env provides methods for connecting to and interacting with a Pokémon Showdown server, removing the need for implementing this functionality. For all training, evaluation against other agents and testing was done using a locally run instance of Pokémon Showdown whilst testing against human players was done using the primary Pokémon Showdown server found at https://play.pokemonshowdown.com/

Multiple files were created for performance testing, each for a different format of testing so that for a specific test, one file could be run. Within the files, all of the agent classes were imported and so agent objects can be used as parameters to the testing functions. Setting parameters such as which agent and number of battles to simulate is done via editing the files rather than user input for efficiency purposes in running files multiple times in testing.

Team

For all battles across all agents the same team was used. This was done to negate the impact of team quality on performance as all agents had the same team and thus the same team quality. The difference would be in how the team was used.

The below table figure 4 shows the team used by the agent. The team is balanced between attack and defence with Tapu Lele and Dragapult focusing solely on inflicting damage whilst the other four have a mix of damage and defensive moves.

Species	Item	Ability	EV Allocation	Nature	IVs	Move 1	Move 2	Move 3	Move 4
Tapu Lele	Life Orb	Psychic Surge	252 SpA / 4 SpD / 252 Spe	+ Spe - Atk	All 31 save 0 Atk	Moonblast	Psychic	Psyshock	Focus Blast
Dragapult	Life Orb	Clear Body	252 SpA / 4 SpD / 252 Spe	+ Spe - Atk	All 31 save 0 Atk	Draco Meteor	Flamethrower	Shadow Ball	Thunderbolt
Landorus Therian	Rocky Helmet	Intimidate	244 Atk / 248 SpD / 16 Spe	+ SpD - SpA	All 31	Earthquake	Knock Off	Defog	Stealth Rock
Rotom-Wash	Heavy Duty Boots	Levitate	252 HP / 4 SpD / 252 Spe	+ Def - Atk	All 31 save 0 Atk	Defog	Thunderbolt	Pain Split	Will-O-Wisp
Ferrothorn	Leftovers	Iron Barbs	252 HP / 4 Atk / 252 Def	+ Def - Spe	All 31 save 0 Spe	Spikes	Knock Off	Leech Seed	Gyro Ball
Heatran	Air Balloon	Flash Fire	252 HP / 4 SpA / 252 SpD	+ SpD - Atk	All 31 save 0 Atk	Lava Plume	Earth Power	Protect	Toxic

Figure 4: Agent's Team

The stats in the EV IV and nature columns are abbreviated as:

Attack -> Atk; Special Attack -> SpA ; Defence -> Def; Special Defence -> SpD; Speed -> Spe

Agent Against Agent Testing

Multiple files were created to carry out testing of agents battling against each other. The same files were used for the Q Learning agent to train which was done against the Max Damage Plan agent. The Max Damage agent was not used in testing or training.

testCrossEvaluate was the first script used for agent-agent testing. The function uses a list of agents and a number of battles and carries out that number of battles between each possible pair of different agents in the list. The script returns and prints a data structure storing the win rate of each agent against the others. The code for this was based on an example the documentation for Poke-env [16].

To make accessing aspects of results beyond overall win rate easier, **data_collection** and **test1v1** were made. Test1v1 is similar to cross evaluate but only carries out one battle between the agents provided in the list. Data_collection calls test1v1 in a for loop to carry out multiple battles for one pair of agents but this approach allows results of each battle to be written to a file after each battle.

data_collection_rr is a script that was used to get the majority of the results. This script is similar to data_collection but on each iteration, it ran test1v1 on multiple pairs of agents, allowing for a full result set to be obtained without having to manually change the pair of agents to get the next set of results.

Agent Against Human Testing

Two scripts were written to facilitate agents battling against humans. The first, **testChallengeMe** will create an object of the set agent class and will issue a battle request to the set username on the locally run instance of Pokémon Showdown. A human logged in to that account can accept the request for a battle. This script was not used to collect any data and was instead used in development to test agent functionality.

The script **testLadder** was used to test against human players on the main Pokémon Showdown server. Pokémon Showdown allows users to battle either by challenging a specific player or by being matched up automatically against other players. testLadder uses the latter of these options with a given agent class and number of battles.

Each agent used its own account to battle so the statistics of the account would be solely associated with one agent.

Results Collection

For the agent against agent battles, the result of the battle is stored in a file, consisting of the agent, either win or loss and the utility of the final recorded state in the battle using utility functions from the Q Learning agent. The Q Learning agent also writes to a file to log changes made to Q values, recording the index of the state, the index of the action and the previous and new utility values.

Results

Agent against Agent Results

The following table shows the results of the agents battling against each other

Winners \ Losers	Max Damage Plan	Q Learning	Monte Carlo	Hybrid Turn Switch	Hybrid HP Switch	Hybrid Info Switch
Max Damage Plan		95.98%	97.59%	100%	93.55%	100%
Q Learning	4.02%		14.91%	22.58%	48.28%	42.86%
Monte Carlo	2.41%	85.09%		41.18%	76.67%	68.97%
Hybrid Turn Switch	0%	77.42%	58.82%			
Hybrid HP Switch	6.45%	51.72%	23.3%			
Hybrid Info Switch	0%	57.14%	31.03%			

Figure 5: Agent vs Agent Results Table

Each percentage shows the win rate of the agent on the left against the agent along the top, e.g., the Max Damage Plan agent had a 95.98%-win rate against the Q learning agent. Not all agents battled each other an equal number of times, but all pairs battled for at least 2000 battles. These results show the Max Damage Plan agent performed significantly better than any other agent, with all win rates being over 90% and with 100%-win rates against the Hybrid Turn Switch and Hybrid Info Switch agents.

Looking beyond the Max Damage Plan agent, the Monte Carlo Agent performed well in comparison to the other agents with a greater than 50%-win rate against the Q Learning agent, Hybrid HP Switch agent and Hybrid Info Switch agent. The 85.09%-win rate against the Q Learning win rate is also significant.

Amongst the Hybrid Agents, the Turn Switch agent performed beset with a 77.42%-win rate against the Q Learning agent vs 51.72% and 57.14% for the HP Switch and Info Switch respectively and a 58.82%-win rate against the Monte Carlo Agent vs 23.3% and 31.03% for the HP Switch and Info Switch respectively.

The Q learning agent performed the worst, with no win rate being greater than 50% against any agent.

Agent Against Human Results

The results against human players come from the Pokémon Showdown ladder. Each player on the ladder has a rating known as Elo [17]. When a player looks for a battle on the ladder, they are matched with a player of a similar Elo as Elo gives a metric for the skill of a player. Winning battles earns a player Elo with the amount earned being related the Elo of the opponent. Larger amounts of Elo are earned from defeating players with more Elo and less Elo is earned from defeating players with less Elo. This system implements the idea that winning against a more skilled opponent demonstrates more skill than winning against a less skilled opponent. Losing battles results in losing Elo with the amount varying in an inverse way to gaining Elo: losing to high Elo and thus high skill players results in a smaller loss in Elo compared to losing to a lower Elo and thus lower skilled player.

A new player on will have an initial Elo of 1000 which is also the minimum value of Elo a player can have.

Another metric used on Pokémon Showdown is Glicko – X Act Rating Estimate (GXE) [18] which gives the probability of the player being able to defeat any random player on the ladder.

Each agent took part in around 15 battles on the ladder. The below table show the results:

Agent	Win Rate (%)	GXE (%)	End Elo
Max Damage Plan	43.56	39.9	1080
Q Learning	23.94	23.6	1032
Monte Carlo	27.92	25.6	1000
Hybrid Turn Switch	20.75	21.4	1032
Hybrid HP Switch	21.57	23.1	1038
Hybrid Info Switch	22.00	21.4	1065

Figure 6: Agent vs Human Results Table

These results show the Max Damage Plan agent performed best with a win rate of 43.56% and a GXE of 39.9% whilst nearly all other win rates and GXEs are below 25% and all of them are below 30%. The Max Damage Plan agent also has the highest Elo at 1080. However, given than battles can typically reward Elo in the range of 10 to 30 per win, the variation in Elo across agents isn't very significant.

After the Max Damage Plan Agent, the Monte Carlo agent is able to differentiate itself as second best with a win rate of 27.92% and GXE of 25.6%. The Q learning agent performs slightly better than the Hybrid agents with a win rate of 23.94% and GXE of 23.6% and the Hybrid agents perform similarly. However, the difference between the hybrid agents the Q learning agent is not significant.

Agent Win Quality

The utility of the final recorded state of each battle was recorded. The following boxplots show the distribution of utility for the wins and losses of the Q learning, Monte Carlo and Hybrid agents.

Higher utilities signify a better position for the agent, so a high utility win is a more decisive win and low utility win is close win. The recorded utility is the larger value recorded from the two agents in the battle so for a loss, a high utility is a decisive loss, and a low utility is a close loss.

Q Learning Agent

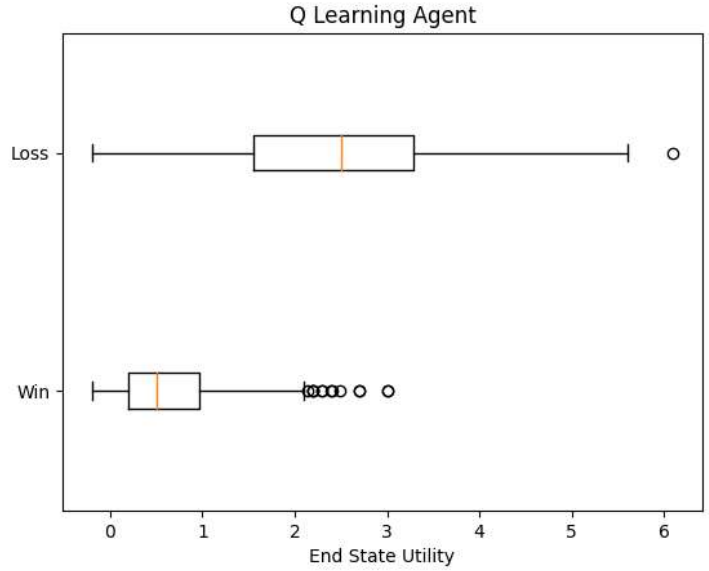

Figure: 7 Q Learning Agent Utility

The results suggest the Q Learning agent performed inconsistently as there is a large range of loss utilities: meaning that loses ranged between being very close and very decisive. The win utilities have a much narrower range focused on the lower end of the graph. The upper quartile is below 1 showing 75% of the wins were very close with only few wins being better but none being as decisive as the losses.

Monte Carlo Agent

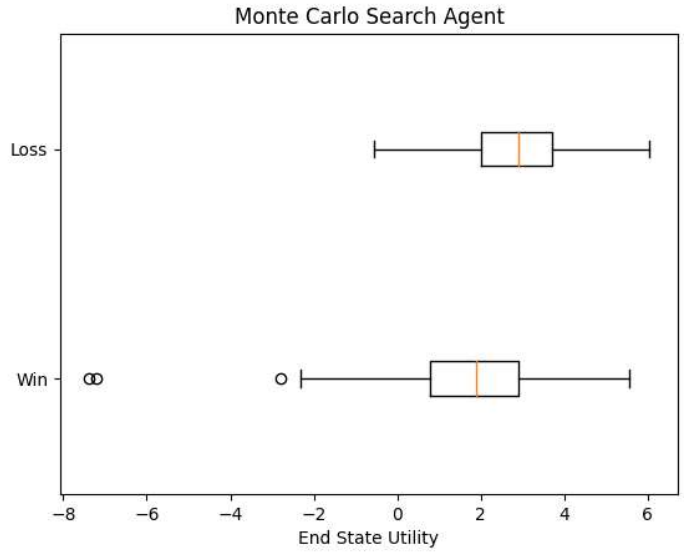

Figure 8: Monte Carlo Agent Utility

Both win and loss show large variation in utility. This shows that the Monte Carlo agent was able to secure some decisive wins and well as decisive losses. The middle 50% of results are greater for losses than wins, showing the Monte Carlo's median losses were more decisive than its median wins.

Hybrid Info Switch Agent

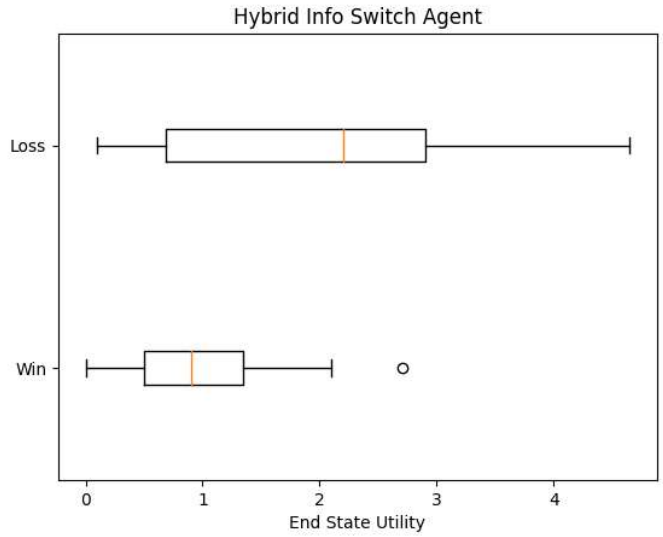

Figure 9: Hybrid Info Switch Agent Utility

The Hybrid Info switch agent shows larger variation in losses compared to wins. 25% of losses range into higher utilities and the second 25% also range highly, suggesting that the agent was more consistent on loss utility at two peaks around 0.5 and 2.5. Wins however were much more consistent and consistently closer.

Hybrid Turn Switch Agent

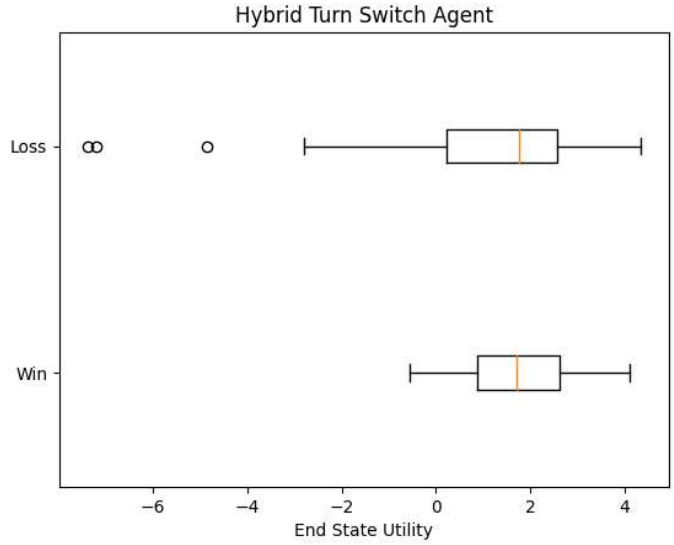

Figure 10: Hybrid Turn Switch Agent Utility

The hybrid turn switch agent was able to reach equally decisive wis and losses at the top end and was able to reach closer losses than wins. The middle 50% for both wins and losses is closer, showing more consistent performance.

Hybrid HP Switch Agent

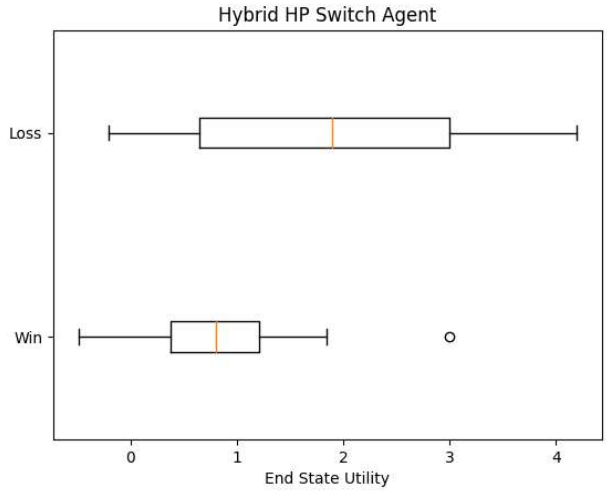

Figure 11: Hybrid HP Switch Agent Utility

The Hybrid HP switch agent had greater variance in losses than wins with similarly close wins and losses and the low end but more decisive losses than wins at the top end. Losses are more varied and more consistently varied across all quartiles whilst the wins less varied in the middle 50%, suggesting that the agent's losses were more evenly distributed across utility than the wins are.

Comparison

All agents aside from the Monte Carlo agent had more varied loss utilities than win rates which sets the Monte Carlo agent aside similarly to the win rate percentages. The Monte Carlo agent was able to get the highest win utilities followed by the hybrid turn switch agent whilst the other agents had similarly lower maximum win utilities. All agents had similar minimums for close losses and close wins. Decisive losses are also similar across the agents.

Q Value Changes

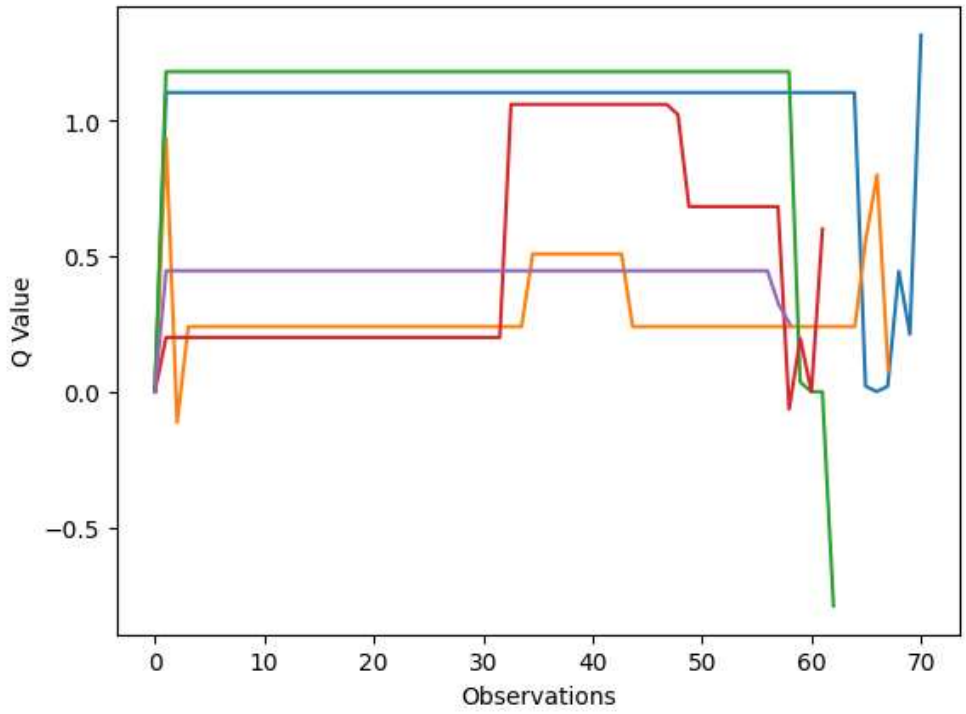

Figure 12: Q Value Changes Graph

The above graph shows the value of action utility across the 5 most updated utility values. All 5 of the values show large changes throughout which suggests that none of the utility values converged for the Q learning agent.

Conclusion

The results show that the Max Damage Plan agent performed the best, being the clear best performer against all other agents and against human players. However, against human players the win rate was still below 50% against low ladder players. It is likely that the Max Damage Agent skewed the loss utility data of the other agents toward being more varied due to decisively defeating whilst the other agents were likely more evenly matched.

Amongst the more complex agents, the Monte Carlo agent performed best, both against the other agents and against human players. This is likely due to the agent accounting for all possible actions the opponent takes, benefiting the agent against less predictable opponents.

The Q learning agent performed very poorly against all opponents and from the action utility value data, this seems to be because the Q learning agent was unable to converge action utility, meaning its performance was closer to a random agent. Pokémon is a difficult game for Q learning due to the stochastic nature as well as the large state space. Performance on the ladder is even more difficult to the larger variety of teams the agent can go up against. The agent rarely encountered the same state across thousands of battles against the same opponent with the same team whereas on the ladder a player may encounter a team only once across thousands of battles which is potentially prohibitive on the agent's success.

Amongst the hybrid agents, the turn switch agent appeared to perform the best with better win rates against the other agents, similar performance against human players and the ability to gain more decisive wins. From observation, battles amongst agents could reach 60 turns which makes switching at ten turns potentially very early. Spending the majority of the battle in the Monte Carlo mode of operation would explain better performance given how well the Monte Carlo agent performed compared to the Q learning agent. It is possible then that the Q learning part was detrimental as none of the hybrid agents could meet or exceed the performance of the pure Monte Carlo agent.

In conclusion combining Q learning and Monte Carlo search in the form of the hybrid agents described in this report does not give greater performance than Monte Carlo search alone but does give greater performance than Q learning alone.

Evaluation

Ultimately, the objective of this project has been successfully achieved. The designed agents were all implemented their performance evaluated against each other as well as human players. The data collected from the simulated battles was able to inform a conclusion about the effectiveness of combining Monte Carlo search and Reinforcement learning in the form of Q learning for Pokémon Battling.

Difficulties were encountered which resulted in changes being needed to be made to aspects of the project. For example, the original team featured moves that inflict damage and prompt a switch. The request for a switch could not be responded to properly by the agents as there was no clear way to distinguish this request from an action request for a new turn. To keep to the schedule and keep focus on implementing and evaluating the agents, these moves were replaced on the team to avoid the problem instead of solving it. Due to all agents using the same team, this change did not affect results even if the modified team was a worse team than the original.

The difficulties and issues encountered, whilst unforeseeable, did not impact the success of the project due to flexibility in the schedule in regard to many tasks being allocated more time than they required.

Future Work

There are many directions future work could take. The Q learning agent's poor performance is a result of the lack of convergence of Q values. Whilst additional training would help, it is possible that the approach may not be suitable for Pokémon battles. Certainly, the knowledge base JSON file could be improved and replaced to speed up battles, allowing for more training in less time.

Expanding upon the Monte Carlo agent's simulation would likely result in greater performance. This can be done by listing more Pokémon and moves to simulate with, as well as accounting for more factors in simulation such as the random 85% to 100% multiplier applied to damage, as well as accounting for items and abilities.

Further work could also be done by testing with different teams and team styles. It is possible that performance could be better with more offensive teams where the full effect and impact on the battle a move has can be more easily modelled and accounted for. The team used included multiple moves related to entry hazards whose value is more difficult to determine due to the long-term nature of their effects.

Both the Q learning and Monte Carlo agents could likely benefit from the implementation of heuristics that could guide exploration and simulation with actions that are likely to always be beneficial in some way.

Overall, there are lots of ways to further work into Pokémon battling with more sophisticated approaches, as well as working with the constantly changing strategies as well as new mechanics adding additional possible actions to the game introduced by the most recent at the time of writing Pokémon Games.

Bibliography

[1] Bulbapedia. (Unknown). Type. [Online]. Bulbapedia. Last Updated: 14 February 2023. Available at: https://bulbapedia.bulbagarden.net/wiki/Type [Accessed 10 April 2023].

[2] Bulbapedia. (Unknown). Held item. [Online]. Bulbapedia. Last Updated: 13 February 2023. Available at: https://bulbapedia.bulbagarden.net/wiki/Held_item [Accessed 10 April 2023].

[3] Bulbapedia. (Unknown). Ability. [Online]. Bulbapedia. Last Updated: 10 February 2023. Available at: https://bulbapedia.bulbagarden.net/wiki/Ability [Accessed 10 April 2023].

[4] Bulbapedia. (Unknown). Move. [Online]. Bulbapedia. Last Updated: 27 February 2023. Available at: https://bulbapedia.bulbagarden.net/wiki/Move [Accessed 10 April 2023].

[5] Bulbapedia. (Unknown). Stat. [Online]. Bulbapedia. Last Updated: 19 February 2023. Available at: https://bulbapedia.bulbagarden.net/wiki/Stat [Accessed 10 April 2023].

[6] Bulbapedia. (Unknown). Pokémon battle. [Online]. Bulbapedia. Last Updated: 13 January 2023. Available at: https://bulbapedia.bulbagarden.net/wiki/Pokémon_battle [Accessed 10 April 2023].

[7] Smogon. (Unknown). Overused SS. [Online]. smogon.com. Last Updated: Unknown. Available at: https://www.smogon.com/dex/ss/formats/ou/#! [Accessed 10 April 2023].

[8] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[9] Silver, David and Hubert, Thomas and Schrittwieser, Julian and Antonoglou, Ioann. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv. [Online]. Available at: https://arxiv.org/abs/1712.01815 [Accessed 28 February 2023].

[10] BBC. (2017). Google AI defeats human Go champion. [Online]. bbc.co.uk. Last Updated: 25th May 2017. Available at: https://www.bbc.co.uk/news/technology-40042581 [Accessed 18th February 2023].

[11] Norström, L (2019), Comparison of Artificial Intelligence Algorithms for Pokémon Battles, Masters Thesis, Chalmers University of Technology, Gothenburg.

[12] Rill-García, R. Reinforcement Learning for a Turn-Based Small Scale Attrition Game

[13] Pikalytics. [Online]. Available at: https://www.pikalytics.com/pokedex/gen8ou [Accessed 19 November 2022].

[14] Sahovic, H (2022), Poke-env, [Source Code], Available at: https://github.com/hsahovic/reinforcement-learning-pokemon-bot

[15] Pokémon Showdown. [Online]. Available at: https://pokemonshowdown.com/ [Accessed 19 November 2022].

[16] Haris Sahovic. (Unknown). Poke-env: A python interface for training Reinforcement Learning pokemon bots. [Online]. Read the Docs. Last Updated: Unknown. Available at: https://poke-env.readthedocs.io/en/stable/ [Accessed 3rd Feb 2023].

[17] Pokémon Showdown. (Unknown). How the ladder works. [Online]. pokemonshowdown.com. Last Updated: Unknown. Available at: https://pokemonshowdown.com/pages/ladderhelp [Accessed 20th Apr 2023].

[18] X-Act (2009) “GXE (GLIXARE): A Much Better Way Of Estimating A Player’s Overall Rating Than Shoddy’s CRE,” smogon.com. Available at: https://www.smogon.com/forums/threads/gxe-glixare-a-much-better-way-of-estimating-a-players-overall-rating-than-shoddys-cre.51169/ (Accessed: March 9, 2023).