PokeBots - Deep Reinforcement Learning vs. Scripting-based Bot
Modern Game AI - Assignment 3 *

Brian Arnesto Sitorus - s3277224, Jasmin Kareem - s3357937, Ramya Tumkur Rameshchandra - s3306593, Srishankar Sundar - s3151298, and Sudarshna Arya Patel - s3259927

Universiteit Leiden, Rapenburg 70, 2311 EZ Leiden, Netherlands

Abstract. This study covers the implementation of two approaches to creating a bot capable of playing Pokemon battles, a turn-based combat scenario that is part of the Pokemon video games. A scripted approach is studied first, relying on game knowledge to drive a battle strategy capable of winning against other bots and players. Secondly, a Reinforcement Learning (RL) approach is taken to drive a bot that learns to play on its own. The bots compete and evaluate on Pokemon Showdown, a battle simulator. The overview and some vocabulary related to Pokemon and the subsequent implementation are first covered, followed by a definition of the problem statement and related work. The implementation and the results obtained over the course of the same are discussed and conclusions are drawn on the results and possible areas of improvement.

Keywords: Pokemon · Deep Reinforcement Learning · Scripting

1 Introduction

Pokemon is a role-playing game franchise developed by Game Freak for various Nintendo consoles. The first pair of games, Pokemon Red and Pokemon Green released in 1996, and since then Game Freak have gone on to release 21 or so pairs of games spread across 8 'generations'. Each generation brings with it updates to graphics, game mechanics and the roster of Pokemon, i.e the characters or creatures that are used in the game's battling system, with the latest generation boasting a roster of 898 Pokemon.

1.1 Pokemon Battles and Related Vocabulary

Pokemon battles are turn-based, and are the central combat mechanic in the Pokemon games. The battles involve two actors, i.e the player and the opponent engaging with up to 6 Pokemon each in their 'party'. At a given moment, a standard Pokemon battle involves one active Pokemon each from either actor. Each

* Universiteit Leiden

Pokemon has a set amount of hit-points (HP), which on depletion renders the Pokemon inactive. A Pokemon has access to up to 4 'moves' at any given moment in order to cause damage to the opponent Pokemon's HP, as well as cause various status effects and alter either Pokemon's statistics mid-battle. Thus, at any given moment, either actor has access to four possible moves and five possible 'switches' to any other Pokemon in that actor's party, where a switch takes up a turn. The battle ends when either of the actors' party has 'whited out', i.e all Pokemon are rendered inactive.

The above is the basic rule-set used for the implementation in this study, although some specifics have been added or changed in the games themselves with each generation.

Fig. 1: Battle interface from Pokemon Emerald(2004) showing 2 possible moves

Both Pokemon and their underlying moves have associated 'types'. There are a total of 18 types as of the latest generation. The types have a rock-paper-scissor-like relationship with each other, with certain types being 'super-effective' against other types and vice versa. These type-matchups are elaborated upon in the coming sections as they are central to one of the approaches used in this study.

1.2 Pokemon Showdown

Pokemon Showdown is a Pokemon battle simulator hosted as a web application. It is free and open-source. Servers can be set up locally, apart from interacting with the official Pokemon Showdown server.

Showdown focuses purely on the battling aspect of the Pokemon games. It allows two actors to engage in a battle with various formats. For the sake of streamlining our approaches, we pick the standard 6 Pokemon-a-side setup for this study. For the implementation, the bots we create interact with a local Showdown server.

It is to be noted that Showdown is one among several battle simulators, each of which provide different battle formats and settings. Showdown also provides

the option to initialize random teams of Pokemon to the agents, which suits the implementation in this study. Team-building in itself is an additional task that is not covered in this study.

2 Problem Statement

The goal of this project is to find an optimal battle strategy for the Pokemon Showdown battle system. In past research [2] [3] [4], Reinforcement Learning methods have often been used to achieve this optimal strategy. In our project in order to find this optimal strategy, we take two approaches to create bots that can play Pokemon battles.

The first approach is to create a scripted bot that works on decisions informed from knowledge of the game. Sequential decisions structured as a decision tree are used to drive a specific strategy that can show enough sophistication to beat out baselines and possibly even other approaches. The second approach uses Reinforcement Learning to facilitate the bot's learning of the game. It uses a Deep Value-Based method with Double DQN to learn to battle effectively. In the training process we use 3 different bots (Random Bot, Maximum Damage Bot, and the implemented Scripted Bot) to see which one performs best. In addition, we wish to assess the performance of the RL bot in general when pitted against the scripted bot.

3 Related Work

When playing against random opponents, Kalose et al. [3] use Reinforcement Learning and a single Q-learning [6] agent to achieve a win rate greater than 65 percent. Using a Minimax-Q agent increases this rate to 90 percent. The authors utilised a discretized state space containing only the hit points, types, and attacks of the active Pokemon in order to create a deterministic simulator with a limited number of Pokemon, types, and attacks. This allowed the authors to restrict the number of Pokemon, types of Pokemon, and attacks.

In contrast to this, the implementations in this study work on the complete roster of 898 Pokemon and the movesets that are present as of Generation 8.

4 Implementation

This section covers implementation specifics, the two main approaches used and their underlying details.

4.1 Development Environment and Packages

- **Language for bot implementation**: Python 3.8
- **Environment**: poke-env 0.4.21
- **Deep Learning API**: keras-rl 2

4.2 Poke-env

Poke-env [5] is a package that acts as a Python interface for interacting with Pokemon Showdown. Thus it abstracts the networking required to connect a bot to a server. Additionally, Poke-env also exposes an OpenAI Gym interface that we interact with to develop the RL bot. Poke-env provides data about the battle, the Pokemon involved, moves, types and other information that is used in the decision making process of the scripted bot, as well as to train the RL bot.

4.3 Action Space

Before describing specifics regarding the scripting and reinforcement learning processes, the action space, i.e all possible actions that can be taken at any turn is discussed below, to give an idea of the breadth of what an agent can do. This is contingent on the battle format being based on the rules put forward by Generation 8, which is the current meta rule-set used at the time of this study.

- Each Pokemon has access to **4 possible moves**
- The current active Pokemon can be switched out for any other Pokemon in the party, taking up a move. Thus this yields a maximum of **5 possible switches** (Since there are a total of 6 Pokemon in a party), provided none of the other Pokemon have had their HP depleted
- Additionally, the agent has access to **one** special move called *Dynamax* that can be used on only one Pokemon during a battle. This move increases the active Pokemon's stats by a vast amount

Thus, there are at least 4 possible moves and at most 10 possible moves for th agent to decide between.

4.4 Scripted Bot

The scripting-based bot relies, as the name suggests, on game knowledge in order to execute a certain battle strategy. Trivial examples of such approaches could include a *Maximum Damage Player*, where the strategy is to always pick the strongest move available for the current active Pokemon. It is interesting to note that, when pit against the initial baseline, i.e a bot that takes random actions at every turn, the *Maximum Damage Player* already has a win-rate of 90 percent.

The scripted bot aims to take a more sophisticated approach as compared to *Maximum Damage Player*, and uses the in-game principle of type-matchups as the basis. Figure 2, illustrates the relationships between the 18 different types. The bot prioritizes switching into Pokemon that have an advantageous move type matchup against the active opponent Pokemon. The entire decision making process is illustrated through the decision tree in Figure 3.

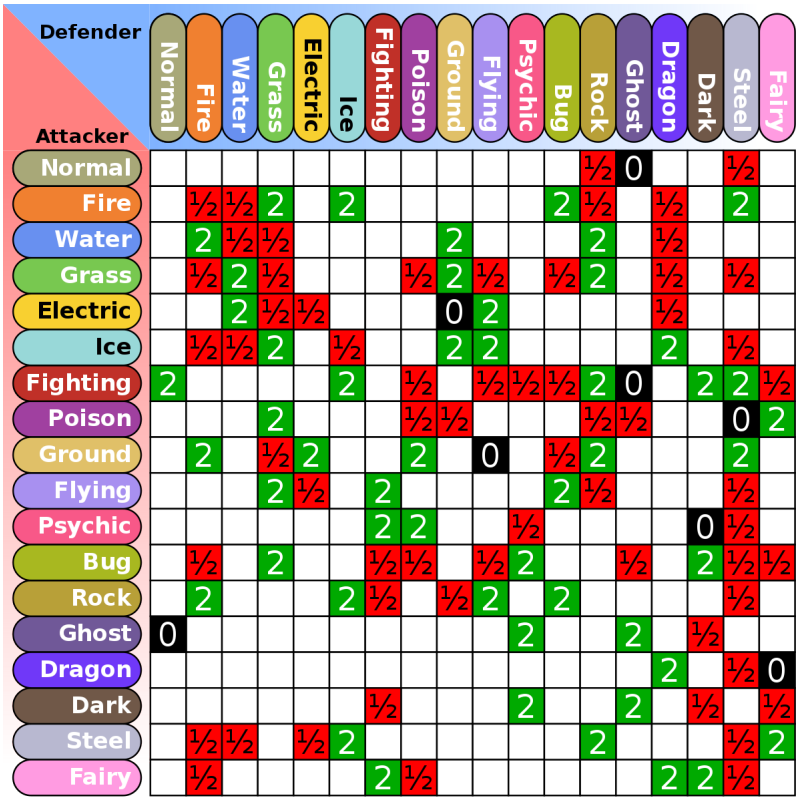

Fig. 2: Type Matchup chart showing which types are effective/ineffective against the other types [1].

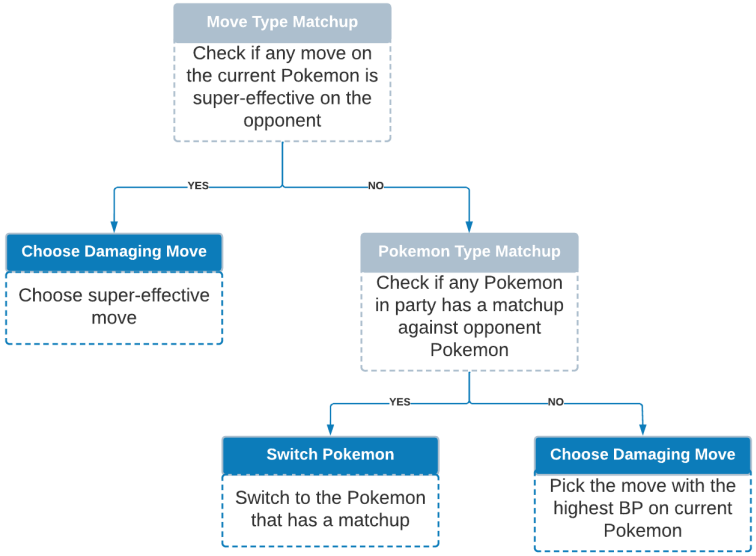

Fig. 3: Decision tree for scripted bot

To elaborate further, the decision tree works in the following way:

- The bot first checks if the current Pokemon has a move with an advantageous type matchup. For example, if the opponent's active Pokemon is *Fire* type, the bot checks if the current Pokemon has any *Water*, *Ground* or *Rock* type moves to play. If it does find one such move it plays it, and plays the strongest such move if there are multiple. (Move strength is governed by a parameter called *Base Power* (BP) associated with a move).

- If the current Pokemon does not satisfy this criteria, it searches for a possible switch to a Pokemon in the bot's party with an advantageous typing. Following the example given in the previous point, the bot searches for Pokemon with the *Water*, *Ground* or *Rock* typing. This is based on the heuristic that a Pokemon of a certain type will always have a move of that very type. Thus, it executes a switch to one such advantageous Pokemon, if possible.

- If this check fails too, as a last resort, the bot plays the strongest move available on the active Pokemon, as a last-ditch attempt to do as much damage as possible to the opponent Pokemon.

4.5 RL Bot

Network Architecture The network is relatively simple, with only one hidden layer. The details are summarized in the table below.

Detail	Value
Number of Hidden Layers, Neurons	1, 64
Activation Function on input and hidden layers	ELU
Optimizer	Adam
Learning Rate	0.00025

Table 1: Network Details

Reinforcement Learning The RL bot uses the OpenAI Gym API exposed by poke-env. The API provides information relating to rewards and observations at each step of an episode. Each step represents a single battle decision, and each battle represents a single episode. For this task, we use **Double DQN** with **keras-rl**. The reward function used is elaborated upon below:

- The agent receives a reward of the percentage of HP damage dealt to the opponent

- The agent receives a reward of 1 if it depletes the HP of the opponent's active Pokemon
- The agent receives a reward of 30 if it wins the battle, i.e it depletes the HPs of all 6 of the opponent's Pokemon
- Conversely, negative actions are rewarded negatively. For example, losing results in a -30 rewards deduction, and so on

Each action is linked to either a *switch* or an *attack*. Additionally, there is an option to *Dynamax* the current Pokemon, which boosts the current Pokemon's stats and takes up a turn. However this is allowed only once in battle.
We implement exploration using the **ϵ-greedy** policy with ϵ set to **0.1**. The model is trained over **10000 timesteps**. These details are summarized in the table below:

Detail	Value
DQN Variant	Double DQN
Discount Factor	0.5
Number of Timesteps	10000
Epsilon	0.1

Table 2: Specifics for Reinforcement Learning

Training We train 3 variants of the RL bot, where each variant has the RL bot being trained against a different opponent. The three opponents include:

- **Random Bot**: Takes random actions at every turn
- **Maximum Damage Bot**: Takes the action with the highest *Base Power* at every turn
- **Scripted Bot**: The scripted bot built on heuristics described in the subsection 4.4 is used for training

The motivation to try different variants arises from the fact the Random Bot alone is too simple an opponent. This is evidenced by the fact that the relatively unsophisticated Maximum Damage Player is able to achieve a win-rate of 90 percent against the Random Bot. Thus we choose tougher opponents to train against to investigate whether this produces more favorable results. The results obtained after training and evaluating these bots are discussed in the following section.

5 Results

The results of pitting the variants of the implemented bots against established baselines and against each other are discussed below.

5.1 Scripting-Based Bot

We see that the scripting-based bot has a win-rate of **90 percent against the Random Bot**, however as established earlier, this is accomplished by the relatively pedestrian *Maximum Damage Bot* too.

When pitted **against the Maximum Damage Bot**, the scripted bot achieves a win-rate of about **75 percent over 1000 games**. Thus we can infer that the strategy of focusing on move and Pokemon type matchups succeeds at some level, against a relatively unsophisticated player.

5.2 RL Bot

In Table 3 below, we illustrate the results of playing each variant against each opponent. The numbers depicted are the win-rates over **1000 games**.

	Opponent			**Score**
Player	Random Bot	Maximum Damage Bot	Scripted Bot	
RL Bot Trained on Random Play	0.98	0.7	0.36	0.68
RL Bot Trained on Maximum Damage Play	0.96	0.7	0.52	0.73
RL Bot Trained on Scripted Play	0.86	0.42	0.18	0.49

Table 3: Performance of 3 variants of RL Bot against Random, Max Damage and Scripted bots

We see that the RL bot trained on Maximum Damage Play comes out on top, with an overall (unweighted) average win-rate of 73 percent, while the least performant variant is the RL bot trained on Scripted play. We also see that the scripted bot performs surprisingly well, and at worst is nearly on par with the RL bot. This is discussed further in the following section.

6 Discussion

There are several points of discussion derived over the course of the implementation and obtaining the results.

6.1 Scripted vs. RL Bot

Based on the results, we can observe that the scripted bot performs well against two of the RL variants, and is nearly on par with the best RL variant with a win-rate of 0.52. While the sophistication of scripting is low, as will be discussed in

the coming sub-sections, this shows that a simple but directed strategy (playing on the basis of type-matchups) is rather effective in general. The inspiration for this strategy arises from the sort of approach a non-competitive average player would take against the AI in the original video games.

It is also important to note that a large part of this result can be attributed to the implementation and training of the RL bot. Further fine-tuning of hyper-parameters could possibly make a much stronger case for the RL bot.

6.2 RL Bot Variants

We observe that the variant trained against Maximum Damage play shows the best performance, while the variant trained on scripted play shows the worst. The initial variant, i.e training on random play is somewhere in between.

The intuition regarding the random bot being too easy or unrealistic of an opponent could be proven true from this result. On the other hand, it is possible that the variant trained on the implemented scripted bot does not perform as well because only training on scripted play could be too steep of a learning task. This results in overall lack of learning.

One further area of work could be investigating performance when the RL bot is trained against mixed opponents, or in sequence. For example, the RL bot could be trained against the Random and Maximum Damage bots initially, and then could eventually be trained on the scripted bot, which is essentially a specific curriculum of ordered tasks.

6.3 Frequency of Switching for the Scripted Bot

A 'switch' takes up a turn that could otherwise have been used to do damage to the opponent. Thus a switch is treated as a weapon to be used only when there is significant advantage in the switch. In the case of the scripted bot, this advantage is viewed as a favorable type-matchup against the opponent's active Pokemon.

To this end, a variant of the scripted bot was also implemented. Instead of simply playing the most powerful move on the active Pokemon's deck when all the checks for possible move and Pokemon type matchups have failed, it would make an additional check for a Pokemon that had a move with a favorable type matchup. For example, if the opponent's Pokemon is a *Fire* type, it would first check for *Water*, *Ground* or *Rock* type moves on the current Pokemon, failing which it would check for *Water*, *Ground* or *Rock* type Pokemon in the party. Failing this check, the normal scripted bot simply plays the most damaging move on the current Pokemon, but the variant then checks for a Pokemon not necessarily *Water*, *Ground* or *Rock* type, but having *Water*, *Ground* or *Rock* type moves, thus still retaining some advantage.

Intuitively, the extra layer of sophistication should have inspired improved performance, however, the extra decisions caused the performance to dip harshly, degenerating to a win-rate of 50 percent against the Maximum Damage Bot.

Upon inspection, it was found that the tree would traverse down to the new check more frequently than was expected, about 20 to 30 percent of the time. Thus it **vastly increased the amount of switches** that the bot made, and consequently this meant the variant spent a lot less time doing damage as compared to the final implementation, and consequently took a lot more damage in return, resulting in more losses. Thus this complication was removed.

This brings up an interesting discussion point about the intention of sophistication in scripting and the discrepancy of said sophistication with results. We infer that more involved decision trees do not automatically mean better play. However, there is definitely room for further sophistication, as discussed in the following subsection.

6.4 Sophistication of the Scripted Bot

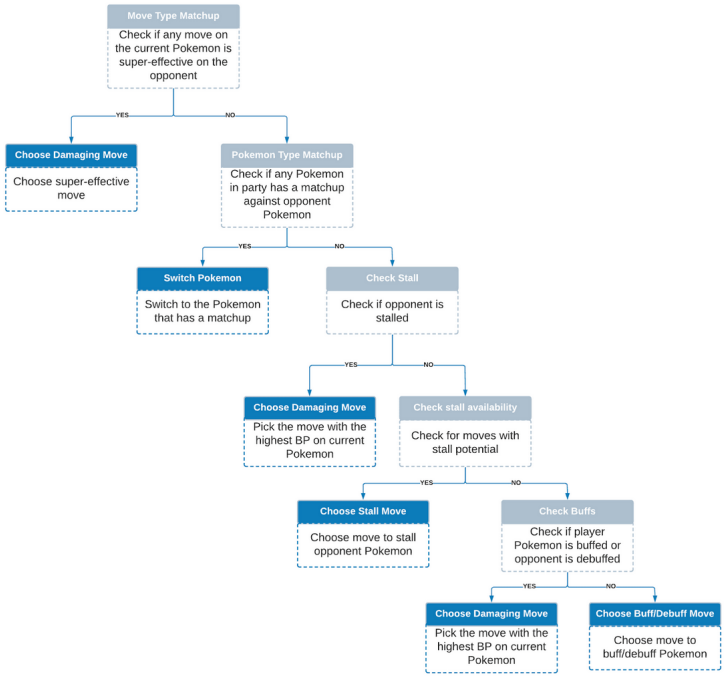

Fig. 4: Possible decision tree for a more sophisticated scripted bot

Although the scripted bot is more sophisticated than the naive Maximum Damage Bot, it could make many more informed decisions after it has failed all the checks in the decision tree. One such example is implementing a strategy for 'stalling', i.e inflicting status effects in case there are no advantageous type matchups. A more complicated decision tree that could be implemented along the same vein is illustrated in Figure 4.

Thus two extra checks could be made, for stalling moves and for de-buffing moves. This could result in more informed play without falling into the trap of switching often. For these reasons there is room for much more sophistication in the scripted approach, which could have an effect on better training and testing for the RL bot.

7 Conclusion

Over the course of this study we cover Game AI concepts with decision trees, heuristics and several concepts under Reinforcement Learning, including the use of Deep Neural Networks, variants of Deep Q Learning, hyperparameter optimization. These relate to lectures 2, 5 and 6 among possible others that were part of the Modern Game AI course at Leiden University.

We study two approaches to creating bots capable of playing Pokemon battles. The scripted approach relies on game knowledge given as a decision tree in order to play, whereas the RL approach learns the game by playing against various opponents. When pit against each other, we see that the best variant of the RL bot has a win rate of 52 percent against the scripted approach. This could be attributed to the training itself, where further tuning of parameters and curricula could help improve performance further. Between the variants of the RL bot, we see that the best performance is exhibited by the bot trained against a Maximum Damage player, an aggressive baseline that naively plays the most powerful move it has access to every single time. The variant trained on the scripted bot itself exhibits low performance. This provides further support to the intuition that a curriculum of ordered tasks could help performance for this task. We also note that there is room for a lot more sophistication for the scripted bot too, and this need not necessarily involve a more complex decision tree.

References

[1] Wikimedia Commons. 2022. File:pokemon type chart.svg — wikimedia commons, the free media repository. [Online; accessed 9-June-2022]. (2022). https://commons.wikimedia.org/w/index.php?title=File:Pokemon_Type_Chart.svg&oldid=639289984.

[2] Dan Huang and Scott Lee. 2019. A self-play policy optimization approach to battling pokémon. In *2019 IEEE Conference on Games (CoG)*. IEEE Press, London, United Kingdom, 1–4. DOI: 10.1109/CIG.2019.8848014. https://doi.org/10.1109/CIG.2019.8848014.

[3] Akshay Kalose, Kris Kaya, and Alvin Kim. 2018. Optimal battle strategy in pokemon using reinforcement learning. *Web: https://web. stanford. edu/class/aa228/reports/2018/final151. pdf*.

[4] Scott Lee and Julian Togelius. 2017. Showdown ai competition. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 191–198.

[5] Haris Sahovic. 2019. Poke-env. (2019). https://github.com/hsahovic/poke-env.

[6] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.