Deep Reinforcement Learning

Final Report

Team Members: Nicholas Garde and Joseph Wagner

12-2-2024
CSCE 642

Team Members: Nicholas Garde 227006946, Joseph Wagner 232002854

Github:https://github.com/Jwagner98/Deep_RL_project
Video Presentation:https://youtu.be/StwDzeWCp1U

Our Plan:
Train an RL Agent to battle against real players on Pokemon Showdown. We aim to conquer this game's complexity and large state and action spaces by finding high-level features to generalize states for use in reinforcement learning algorithms.

For this project, we seek to expand the feature space of the Poke-Env[1] to improve model training and to our model online. Improving model training occurred in 3 steps, first, we identified the feature space which Poke-Env is set up for. Second, we expanded the feature space of Poke-Env. Third, we attempted to improve the training methods for this expanded feature space by increasing training time and finding multiple opponents to play against. After training and benchmarking our RL algorithms we allowed our agent to play online to determine the RL agent's efficiency of play.

Starting Point:
The starting point of our project is to use a git repository called “Poke-Env” that is compatible with OpenAI Gymnasium which lets our agent battle against real players. This environment’s documentation [1] and GitHub link [2] are below in the references section.

Legitimacy:
There are many different competitive pokemon battle formats for our agent to train on, we have chosen the current version *Random Battles* format. This means every game our agent will be given a randomized team of 6 pokemon from a pool of 506 pokemon in *Generation 9 Random Battles*, and our agent’s opponent will also be given a random team of 6. This makes the number of possible starting states,

$$P(506, 6)^2 = 2.65 * 10^{32}$$

as a conservative estimate considering that each of the pokemon provided also comes with a random set of moves, ability, and item.

Pokemon is a game that lies close to both my partner and my heart as it is a game that we’ve played as kids. To train our agent to play the game we used an OpenAI Gymnasium compatible environment. This is no simple task as the number of starting states for *Generation 9 Random Battles* alone is naively 2.65e32+ and there are up to 14 valid actions per state. This complexity is also compounded since the agent lacks complete knowledge of what an opponent's actions could be, like in chess, determining the potential actions of an opponent can become closer to 35+ possible actions. Unlike chess, the true effects of actions are not truly calculable as there is a good amount of stochasticity built into the game. This prevents each game from being deterministic making it a perfect environment to train an RL agent.

Sequence of Events:
In the *Generation 9 Random Battle* format, one random pokemon from each team is chosen and deployed as that team’s active pokemon. This begins **Turn 1** where both players select an action: (naively) one of their active pokemon’s 4 moves or a switch to one of their 5 bench pokemon. These two actions are performed in the order: Switches (if both switch, the higher speed stat active pokemon will switch first), Terrastallization Activation, and finally Moves speed stat ordered, however many moves can have a priority stage meaning they occur before moves in a lower priority stage, and speed ordered when in a tied priority stage. Once both actions have been attempted in sequence, the next turn begins and players select what action to take next.

State Space:
The state space will be defined by the Poke-Env environment turning the game into an object that can be used for predicted action/reward calculation by collecting the observations in a given state. The States of our RL-agent depend on both our pokemon and the opponents'.

Observations:
These are the observations that we gather, including their bounds and why we believe them to be useful.

- **Game:**
  - **Field Status - 13 one-hot field statuses [0,255]**
    - Field conditions that may affect both active pokemon and their moves, includes the turn they were activated.
  - **Weather - 9 one-hot weather statuses [0,255]**
    - Weather conditions that may affect both active pokemon and their moves, includes the turn they were activated.
  - **Turn - [0,255]**
    - Current turn.
- **Agent Side:**
  - **Terrastallization Used - [0,1]**
    - Boolean for bonus terrastallization action that changes the active pokemon's type and increases outgoing damage.
  - **Hazards - 11 one-hot hazard statuses [0,255]**
    - Offensive and defensive side elements that provide stat boosts/drops, deal damage, or apply status.
- **Active Pokemon on Agent Side:**
  - **Move Power - 4 moves [-1,3]**
    - "Base Power" of a given move, correlates to damage dealt.
  - **Move Type Multiplier - 4 moves [0,4]**
    - Moves have additional multipliers based on the typing of the pokemon they hit.
  - **Remaining Health - [0,1]**
    - Fraction of health remaining.
  - **Stats - 7 stats [-6,6]**
    - HP, Attack, Special Attack, Defense, Special Defense, Speed values of a given pokemon that influence how they deal/recieve damage and turn order.
- **Bench Pokemon on Agent Side:**
  - **Remaining Health - 5 bench pokemon[0,1]**
- **Opponent Side:**
  - **Terrastallization Used - [0,1]**
  - **Hazards - 11 one-hot hazard statuses [0,255]**
- **Active Pokemon on Opponent Side:**
  - **Remaining Health - [0,1]**
  - **Stats - 7 stats [-6,6]**
- **Bench Pokemon on Opponent Side:**
  - **Remaining Health - 5 bench pokemon[0,1]**

Action Space:
In this format, a player has up to 14 action choices: 4 attacks, Terrastallization + 4 attacks, 5 switches
Attack:

Every pokemon will have a choice of 1 out of 4 attacks

Terrastilize (Tera):

Current format allows for 1 **Tera** action per game per player that will change a pokemon's typing to a single predetermined type and increases damage using that type by 1.5x (Changing to a type that a pokemon was originally raises this to 2x), Then **Attacking**: with 1 of 4 moves

Switch:

An agent can also use a turn to swap out their active pokemon for one on their bench, who will instead be affected by their opponent's turn and be able to act in the following turn

Rewards:

$$R = (Team_{Remaining\ Health}) - 2 * (Team_{\#\ Fainted}) - (OpTeam_{Remaining\ Health}) + 2 * (OpTeam_{\#\ Fainted}) + \frac{25}{Turn}$$

$$R \mathrel{+}= 30 \text{ on win}$$

$$R \mathrel{-}= 30 \text{ on loss}$$

The reward function depends on pokemon health, fainted pokemon, current turn, and win/loss. These are the major changes from state to state that gauge how a player is doing in that game, and we wanted to incentivize a faster and more decisive game so we added an inverse relationship with turn number since typical games last from 20~40 turns.

Algorithm selection:
We began our research with Jett Wang's and others implementation [3,4] of a similar reinforcement learning agent, where he designed a model on an MCTS using an Actor-Critic network trained using PPO and leveraged self-play on an older version of the game. During the class reading's and literature review we explored several different algorithms that we could use in order to train our RL Agent. Initially interested in any algorithm which had high performance metrics such [4-10] such as A3C and MuZero.However, we had to trim our algorithms to handle only discrete action spaces eliminating A3C. This led us to be primarily interested in A2C and DQN[5,6,7]. We designed an observation space similar to his to gather relevant battle information and made a DQN model [5] that uses an experience replay buffer and gradient clipping as well as a 1-step Actor-Critic model [6] for comparison.

Training:

Baseline:

This algorithm written by Hasvoc[1] trains a Simple RL agent on a limited number of feature against a random player for *n* timesteps to allow for the agent to observe the cost and reward.

Alg1:

We designed training to allow a model to play against a random player for *n* timesteps to allow it to gain experience performing actions and observe effects from the opponent's moves.

Alg2:

We designed training to allow a model to play against a random player for *n* timesteps, then max damage player for *n* timesteps, and finally play against a simple heuristic player for *n* timesteps. This allows our agent to train on the random player for action space discovery, then against max damage player to start developing strategy against constant incoming damage, and against simple heuristic player that weighs the state and decides an action based on apparent danger or advantage.

Results:

We perform evaluation by having a model play 100 games against Random Player, 100 games against Max Damage Player, and 100 games against Simple Heuristics Player.

The Random Player performs poorly as it will often do nothing to the opponent and take drastic net losses on many turns that are difficult to recover from. A moderate experienced human can beat this 90~95% of the time.

The Max Damage Player always selects the move it expects to deal the most damage and ignores any set-up or multi-turn strategy. This type of player could be overwhelming, but overcome by utilizing type matchups and stats to mitigate damage received while dealing more damage out. A moderate experience human can beat this ~75% of the time, however this is a hesitant estimate. Once strategy is involved, the stochasticity of the game can result in unfair disadvantages that could favor one player's team over the other.

The Simple Heuristic Player uses a deterministic strategy based on the state of the game. This allows it to select actions that deal good amounts of damage while also considering defensive actions such as switching and utilizing moves that don't deal damage or don't deal large amounts of damage for their other benefits. A moderate experienced human can beat this ~65% of the time, but again an estimate. Its strategy allows for long term advantages that could prove difficult to beat.

**Additional Note: Joe as a low-moderate experience player did some games against the latter 2 players. He had about a 20% winrate against the Max Damage Player and 67% winrate against Simple Heuristic.

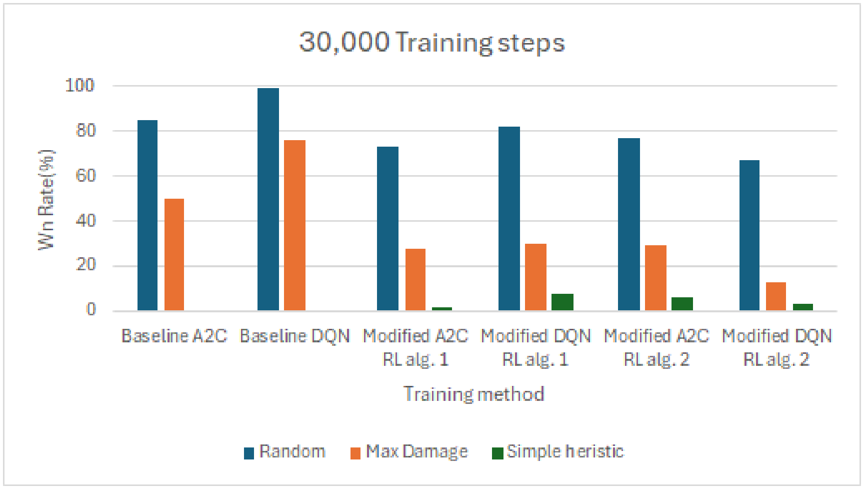


Figure 1: Increased observation space results compared to baseline

Figure 1 Above shows the overall win rates for the Baseline, Modified RL Alg.1, and Modified RL Alg. 2. This Figure shows that the Original Baseline Model

A2C Analysis: ***Fig. 2***
A2C models perform significantly worse than DQN models, though I would be interested to see how it performs after 500k, 1M, and 2M training timesteps. The A2C models show capabilities to perform intelligent actions, however, not exceeding 80% winrate against a random player is a sign of poor performance. Against players with a strategy, our A2C models could not keep up.

We chose to use A2C as one of our algorithms because there are many "micro-strategies" in this domain that involve the vast amount of pokemon our agent could play with. A2C is able to converge on stochastic optimal policies which seems perfect for the domain. A model can approximate simple policies for 2~3 turns to attempt to increase reward before being forced to switch pokemon and use a different strategy.

DQN Analysis: ***Fig. 2***
DQN models perform well, at 250k training timesteps (per opponent) exceeding 90% winrate against Random Player. This is a great benchmark for an RL agent as it shows intent to win against a player with no intent. Against Max Damage Player winrate sits around 60%. This shows that the model can keep up with the momentum of a game against a player that tries to reduce its health to 0. 60% is a fair winrate given the stochastic nature of the game and general team balancing, but nothing ground breaking. Against Simple Heuristic Player, our DQN agent managed above 20% winrate. This shows progress, but the agent would not perform well against live players.

We chose DQN as one of our algorithms because of the algorithms efficiency and effectiveness at learning state-values. We need our agent to be capable of discerning its standing in a game and perform actions to improve it to beat its opponent. The nature of this game encourages taking an advantage and keeping it as shown by its high win rate below.

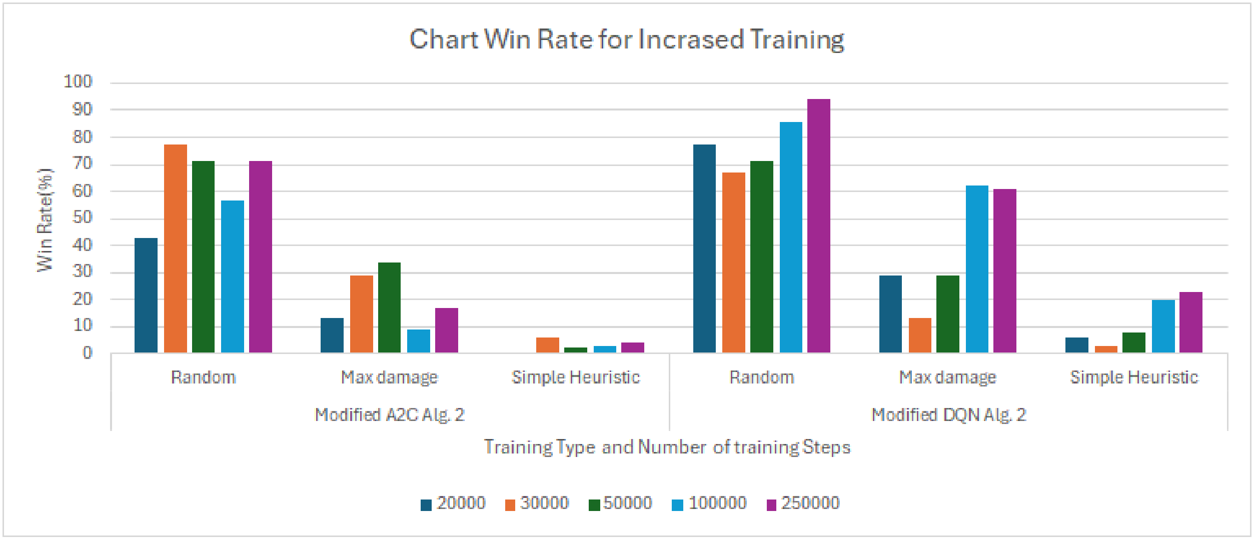


Figure 2: Comparison between the two different RL algorithms up to 250,000 timesteps of training against all 3 opponents in sequence (up to 750,000 timesteps total)

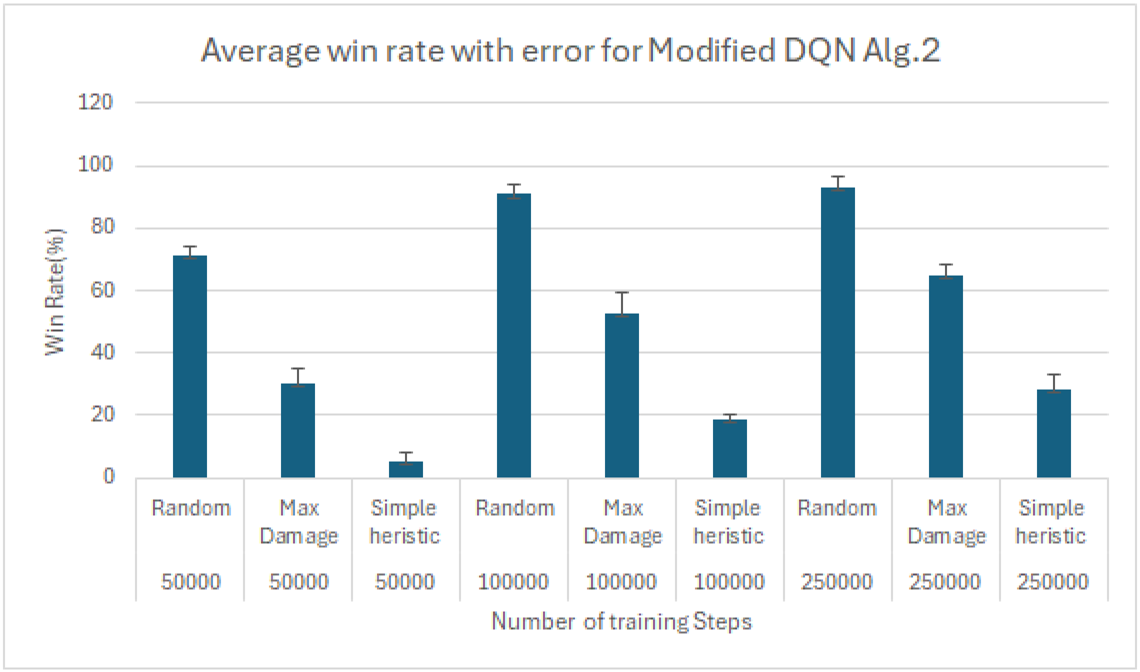


Figure 3: Error analysis of Modified Rl Alg.2

The **Figure** above should the error averaged between 5 experiments, these experiments show the change in win rate for the Modified DQN Alg. 2 which utlized multi step training.

Online Play results:

To watch us play online refer to the following video's

https://youtu.be/996Ftd9mtTM ,https://youtu.be/pKthlW1N1TI, https://youtu.be/BTdrwvnS5Yg

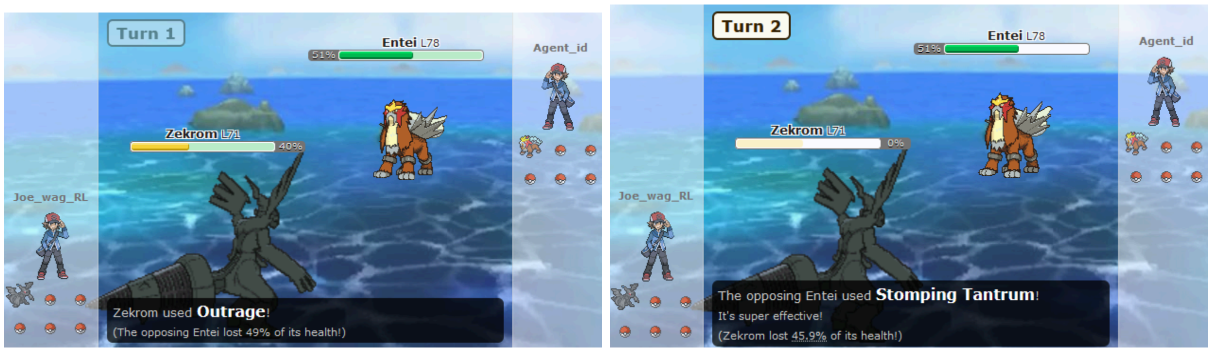


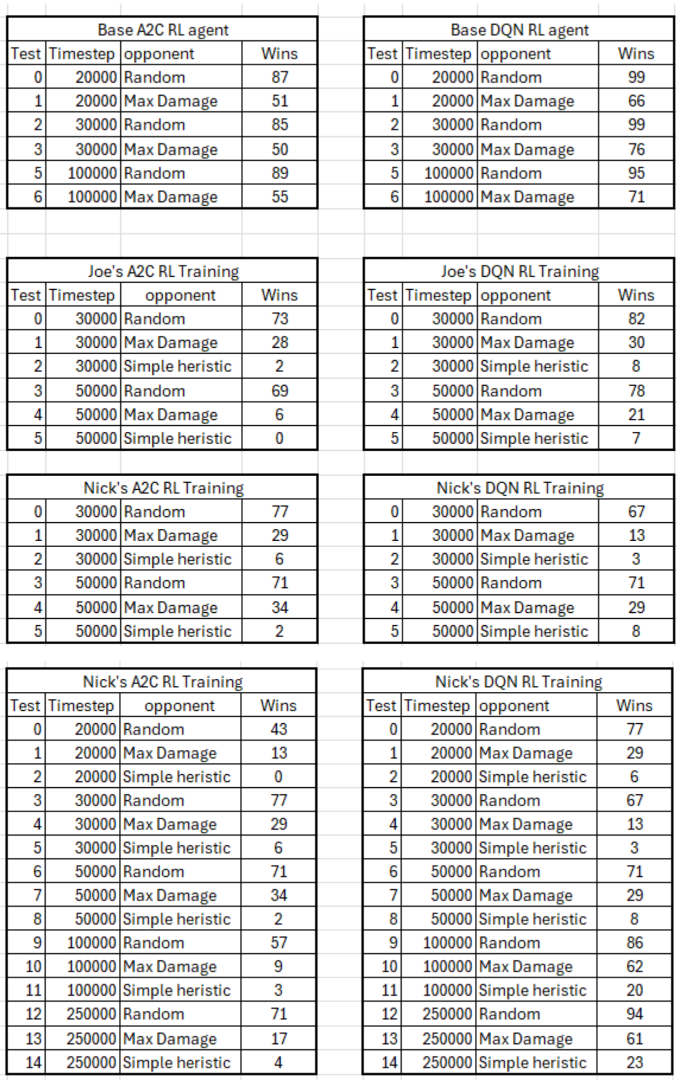
Figure 3: Show's screenshots of of us battling our RL agent https://youtu.be/pKthlW1N1TI

Testing our best model on experts, Joe found it to be a very challenging opponent and did not win any games against it. Nick however has played this game for a while and is knowledgeable about the test format. Our model was unable to win any games against Nick, though it demonstrated intelligence by using terrastallization at good points in the game and performed several switches in disadvantageous positions that resulted in a better outcome than attempting to attack. We consider it a low-mid rank player that could improve its game with more (and perhaps online or self-play) training.

Deficiencies:
Our final model was 250k DQN using algorithm 2 training. We chose this model on its performance and showed signs of playing intelligently in an incredibly complex environment. Unfortunately, we were unable to create larger models for testing due to time constraints and environment issues. When attempting larger trained models we began encountering unexpected errors, namely model deadlock and server disconnects deep into training sessions. There are also even more features that we believe would prove beneficial like more typing data, and a database of: moves, items, and abilities. Our original goal was to have our agent play against many players on the *Gen9RandomBattles* ladder and grade it based on elo as it played. This proved difficult as we got banned very quickly and instead had to test our model on experts.

Conclusion:
In conclusion, we expanded the feature space of the basic Poke-Env model and successfully achieved online play. As a result of expanding this feature space we were unable to improve upon the original Poke-env Test players in random and max test metrics. However, we identified the feature's and training metrics that could be manipulated to make an even more successful RL-agent.

Appendix



Base A2C RL agent			
Test	Timestep	opponent	Wins
0	20000	Random	87
1	20000	Max Damage	51
2	30000	Random	85
3	30000	Max Damage	50
5	100000	Random	89
6	100000	Max Damage	55

Base DQN RL agent			
Test	Timestep	opponent	Wins
0	20000	Random	99
1	20000	Max Damage	66
2	30000	Random	99
3	30000	Max Damage	76
5	100000	Random	95
6	100000	Max Damage	71

Joe's A2C RL Training			
Test	Timestep	opponent	Wins
0	30000	Random	73
1	30000	Max Damage	28
2	30000	Simple heristic	2
3	50000	Random	69
4	50000	Max Damage	6
5	50000	Simple heristic	0

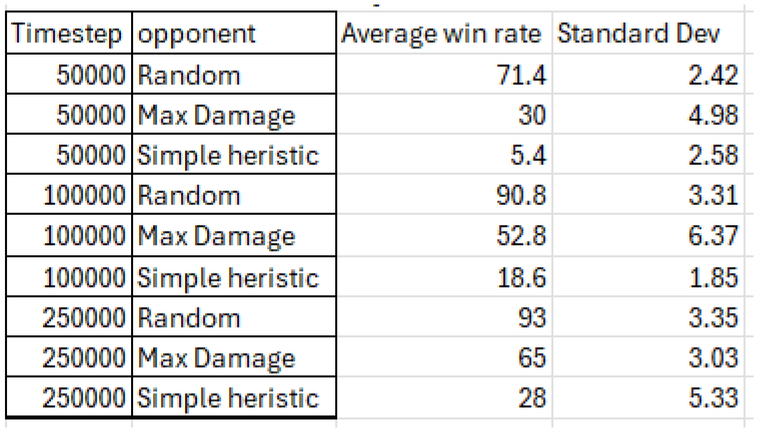
Joe's DQN RL Training			
Test	Timestep	opponent	Wins
0	30000	Random	82
1	30000	Max Damage	30
2	30000	Simple heristic	8
3	50000	Random	78
4	50000	Max Damage	21
5	50000	Simple heristic	7

Nick's A2C RL Training			
0	30000	Random	77
1	30000	Max Damage	29
2	30000	Simple heristic	6
3	50000	Random	71
4	50000	Max Damage	34
5	50000	Simple heristic	2

Nick's DQN RL Training			
0	30000	Random	67
1	30000	Max Damage	13
2	30000	Simple heristic	3
3	50000	Random	71
4	50000	Max Damage	29
5	50000	Simple heristic	8

Nick's A2C RL Training			
Test	Timestep	opponent	Wins
0	20000	Random	43
1	20000	Max Damage	13
2	20000	Simple heristic	0
3	30000	Random	77
4	30000	Max Damage	29
5	30000	Simple heristic	6
6	50000	Random	71
7	50000	Max Damage	34
8	50000	Simple heristic	2
9	100000	Random	57
10	100000	Max Damage	9
11	100000	Simple heristic	3
12	250000	Random	71
13	250000	Max Damage	17
14	250000	Simple heristic	4

Nick's DQN RL Training			
Test	Timestep	opponent	Wins
0	20000	Random	77
1	20000	Max Damage	29
2	20000	Simple heristic	6
3	30000	Random	67
4	30000	Max Damage	13
5	30000	Simple heristic	3
6	50000	Random	71
7	50000	Max Damage	29
8	50000	Simple heristic	8
9	100000	Random	86
10	100000	Max Damage	62
11	100000	Simple heristic	20
12	250000	Random	94
13	250000	Max Damage	61
14	250000	Simple heristic	23



Timestep	opponent	Average win rate	Standard Dev
50000	Random	71.4	2.42
50000	Max Damage	30	4.98
50000	Simple heristic	5.4	2.58
100000	Random	90.8	3.31
100000	Max Damage	52.8	6.37
100000	Simple heristic	18.6	1.85
250000	Random	93	3.35
250000	Max Damage	65	3.03
250000	Simple heristic	28	5.33

References

[1] **“ENV: A Python Interface for Training Reinforcement Learning Pokémon Bots.” *Poke*, poke-env.readthedocs.io/en/stable/. Accessed 30 Sept. 2024, https://poke-env.readthedocs.io/en/stable/.**

[2] Hsahovic. “HSAHOVIC/Poke-Env: A Python Interface for Training Reinforcement Learning Bots to Battle on Pokemon Showdown.” *GitHub*, github.com/hsahovic/poke-env. Accessed 30 Sept. 2024, https://github.com/hsahovic/poke-env.

[3] Wang, Jett. "Winning at Pokémon Random Battles Using Reinforcement Learning." *Dspace@MIT*, 2022, https://dspace.mit.edu/handle/1721.1/153888.

[4]Sarantinos, Panagiotis. "Teamwork under Extreme Uncertainty: AI for Pokémon Ranks 33rd in the World." *arXiv*, 27 Dec. 2022, https://arxiv.org/pdf/2212.13338.

[5] Nair, Ashish, et al. "Massively Parallel Methods for Deep Reinforcement Learning." *arXiv*, 23 July 2015, https://arxiv.org/abs/1507.04296.

[6] Mnih, Volodymyr, et al. "Human-level Control through Deep Reinforcement Learning." *Nature*, vol. 518, 2015, pp. 529-533, https://www.nature.com/articles/nature14236.

[7] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed., MIT Press, 2018.

[8] Mnih, Volodymyr, et al. "Asynchronous Methods for Deep Reinforcement Learning." *Papers with Code*,2016 https://paperswithcode.com/paper/asynchronous-methods-for-deep-reinforcement#code.

[9] van Hasselt, Hado, et al. "Bigger, Better, Faster: Human-level Atari with Human-level Efficiency." *Papers with Code*, 2023 https://paperswithcode.com/paper/bigger-better-faster-human-level-atari-with.

[10] Schrittwieser, Julian, et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model." *Nature*, vol. 588, 2020, pp. 604-609, https://www.nature.com/articles/s41586-020-03051-4.