DarkrAI: a Pareto ε-greedy policy

Improving Pokémon AI Traning With NSGA-II

Simone Alghisi Samuele Bortolotti Massimo Rizzoli
Erich Robbi

University of Trento

August 25, 2022

UNIVERSITÀ DI TRENTO

Pokémon

Pokémon uses a turn-based system: at the start of each turn, both sides can choose to attack, use an item, switch the Pokémon for another in their party. The Pokemon who strikes first is determined by the Move's Priority and the Pokémon Speed. Each Pokémon uses moves to reduce their opponent's HP until one of them faints, i.e. their HP reach 0. If all of a player's Pokémon faint, the player loses the battle.

Figure 1: Pokémon battle

Reinforcement Learning

Reinforcement learning (RL) is an area of Machine Learning where an agent receives a reward based on the action it has performed. Actions allow the agent to transition from a state to another. The final objective is to learn a policy to reach a terminal state with the best reward achievable.

Deep Q-Learning

The reinforcement learning technique we have employed is called *Deep Q-Learning*, which maps input states to a pair of actions and Q-values using an Artificial Neural Network. *Q-Learning* is based on the *Q-function*, namely $Q : S \times A \rightarrow R$, which returns - given a state-action pair $(s, a \in S \times A)$ - the expected discounted reward $(r \in R)$ for future states.

UNIVERSITÀ DI TRENTO

NSGA-II

NSGA-II is a Evolutionary Algorithm that allows to produce *Pareto-equivalent* (or non-dominated) solutions of a multi-objective optimisation problem.

General idea

The idea is that, given that the search space is very big - there are 10^{354} different ways a Pokémon battle can start, and each turn has at most 306 different outcomes (and only for a single player) - we would like to positively bias our model with a controlled search, removing particularly useless moves, i.e. consider for the most Pareto-equivalent solutions.

UNIVERSITÀ DI TRENTO

Genotype representation

Generally, in a Pokémon battle two actions are possible, i.e. performing a move or a switch. Moreover, depending on the type of battle, it may be necessary to specify the target of the move. To encode such a thing, we came up with the following genotype: each Pokémon is represented using two genes, i.e. action and target (optional) (a, t). The whole genotype tells us who is going to perform what on whom.

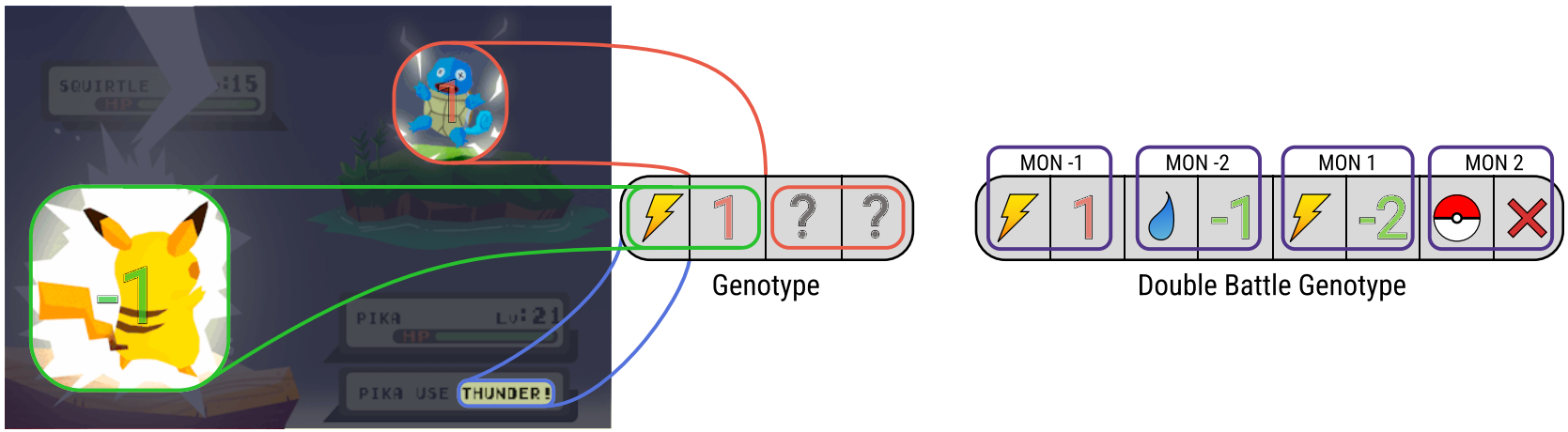

Figure 2: Genotype representation

Genetic operators - Mutation

Mutation is performed for each gene in a genotype with probability $\mathbb{P}_m = 10\%$: both the action and the target may be mutated, meaning that it is possible to go from a move to a switch (and vice-versa).

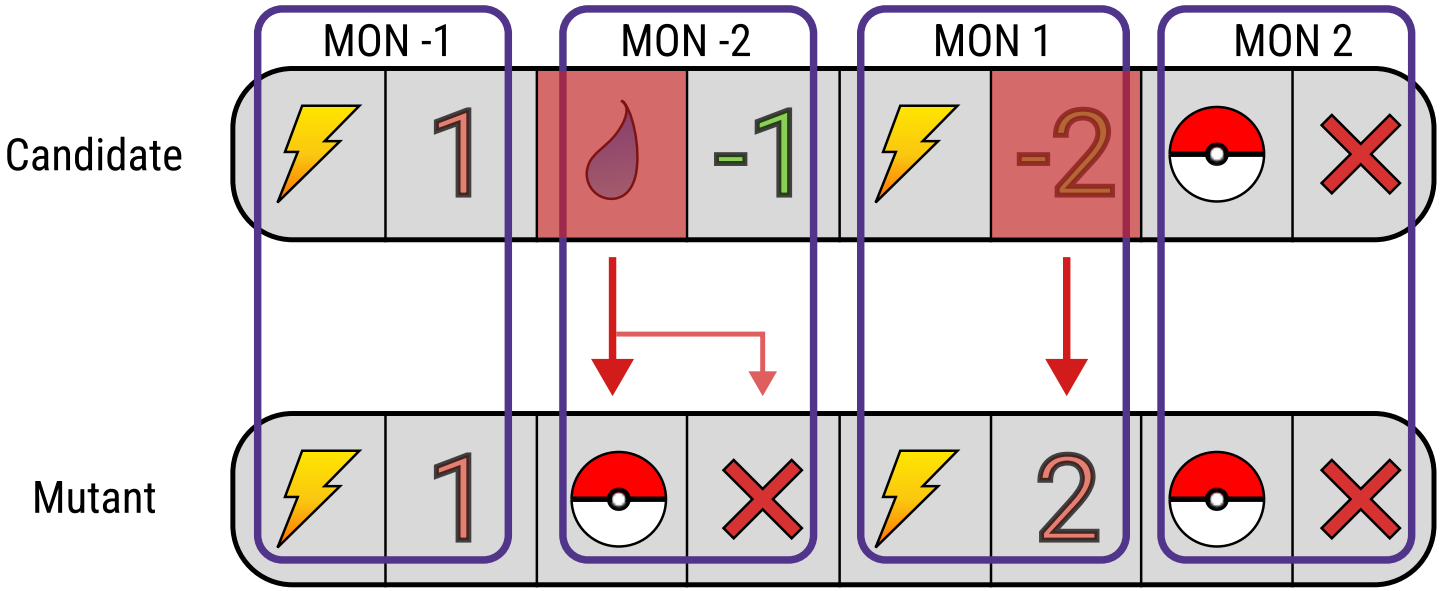

Figure 3: Mutation

Genetic operators - Recombination

Instead, we used Uniform Crossover in a particular way: given that each Pokémon is represented by a valid (a, t) pair, we perform crossover by selecting the whole pair from one of the parents to avoid inconsistencies. Furthermore, crossover is performed with $\mathbb{P}_c = 100\%$, and $\mathbb{P}_{bias} = 50\%$ (i.e. the bias towards a certain offspring).

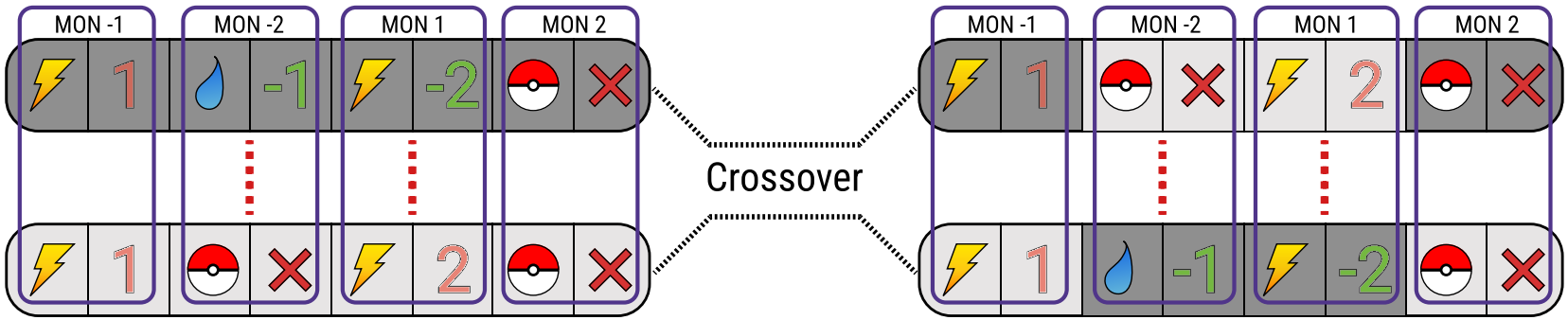

Figure 4: Recombination

Objective & Optimisation

Concerning the *Pareto front* we have considered four variables with the following optimisation problem:

$$\underline{x} = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4 \quad \text{where} \quad \mathbb{R}^4 = \{(x_1, x_2, x_3, x_4) : 0 \leq x_1, x_2, x_3, x_4 \leq 100\}$$

where x_1 is the damage dealt by the ally Pokémons to the opponents, x_2 is the damage dealt by the opponents' Pokémons to the allies, x_3 is the health points remaining of the player's Pokémons and x_4 is the health points remaining of the opponent's Pokémons.

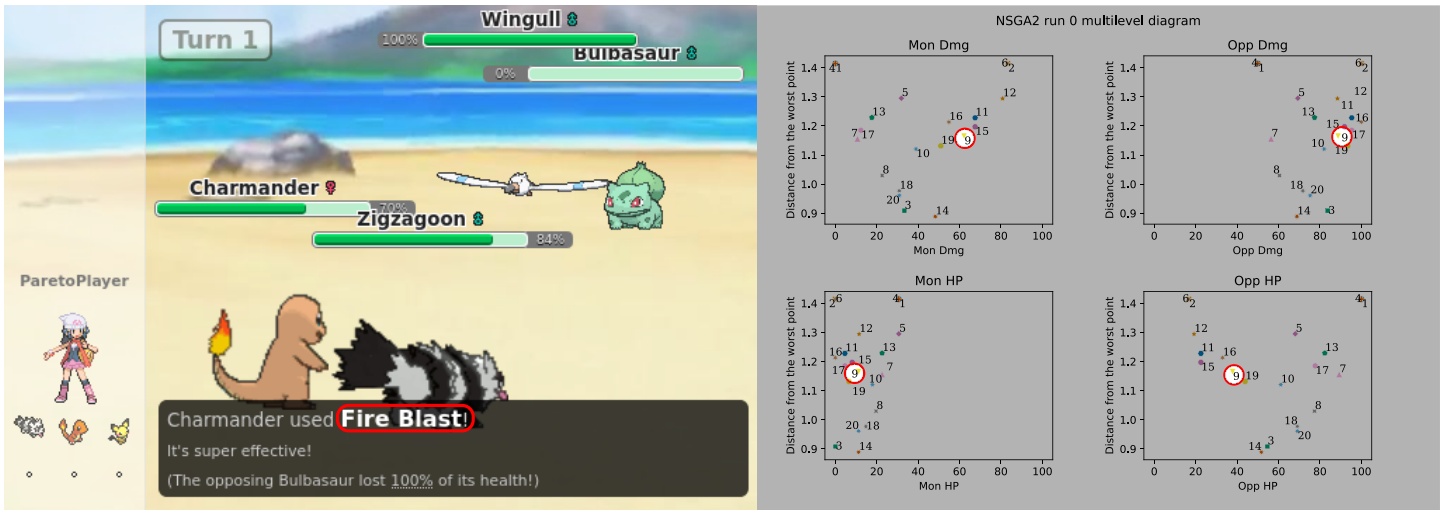

Figure 5: Recombination

Architecture details

The agent architecture is a four-layer deep *Multilayer Perceptron (MLP)*, which employs *ReLU* as activation function. In particular:

- input and output layers size depend on the type of battle the network is facing (e.g. a 4 VS 4 battle implies a size of 244 input neurons);
- two hidden hidden layers of size 256 and 128, respectively.

UNIVERSITÀ DI TRENTO

Players

The standard agent uses a simple ε-greedy policy:

- it starts from a probability $\mathbb{P}_r = 1.0$ to perform a random action;
- it linearly decreases to $\mathbb{P}_r = 0.1$ in the first 40% of the training;
- for the remaining 60% of the training it linearly decreases to $\mathbb{P}_r = 0.01$.

ParetoPlayer

ParetoPlayer embeds the Pareto search of non-dominated moves:

- it performs a random move chosen from the ones returned by *NSGA-II* with 70% probability;
- a completely random one with 30% probability.

UNIVERSITÀ DI TRENTO

Program structure

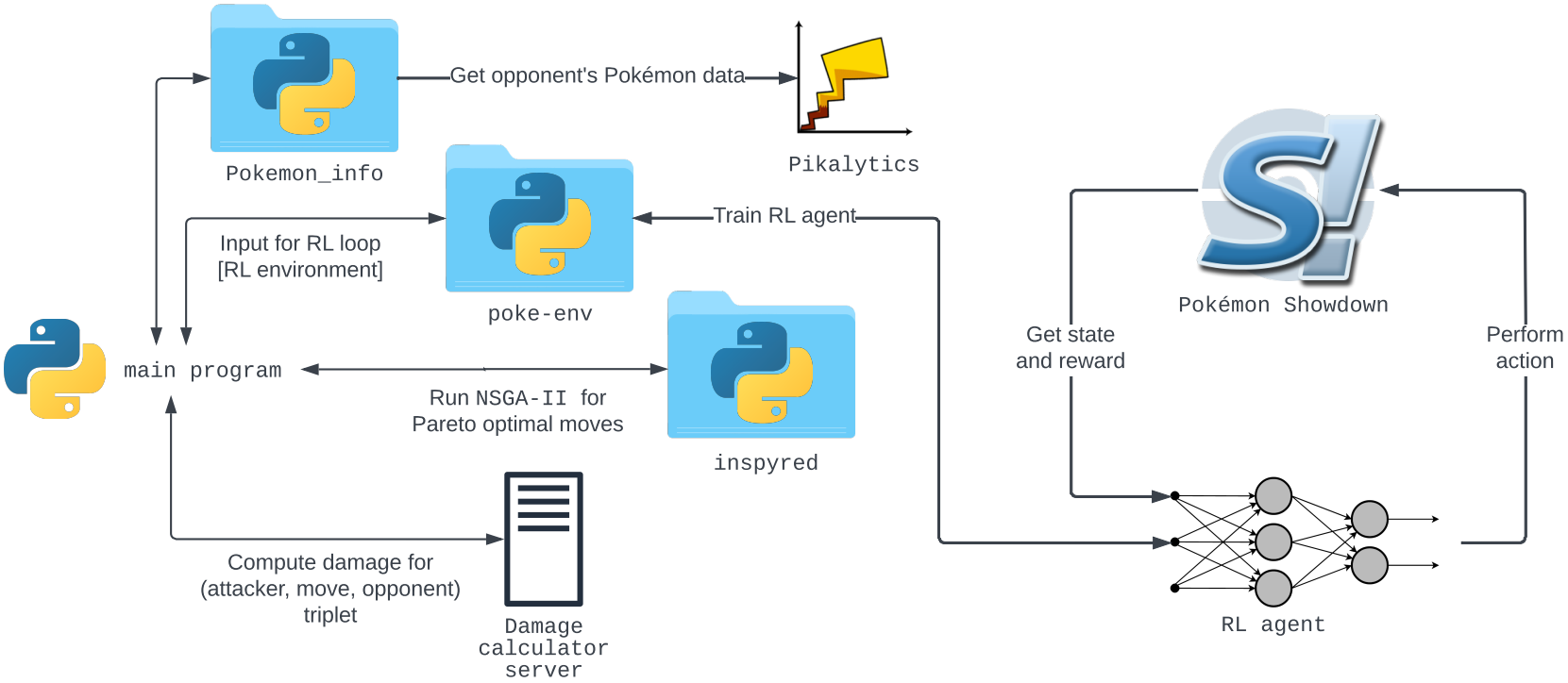

Figure 6: Program structure

Experiments

All agents were trained by having them fight against *MaxDamagePlayer*, i.e. a bot which always chooses the combination of moves that deals the highest amount of damage.

Several situations were considered, such as:

- 2 VS 2 battle with static teams;
- 2 VS 2 battle with the opponent team sampled randomly from a pool of possible Pokémons.

UNIVERSITÀ DI TRENTO

Statistical tests

We have tested both the normality and the statistical significance of the proposed solution with the employment of the following graphical and analytical tools:

Normality

- Quantile-Quantile plot
- Shapiro-Wilk test
- Kolmogorov-Smirnov nomality test

Statistical significance

- Box plot
- t-test
- Wilcoxon rank mean test

UNIVERSITÀ DI TRENTO

Empirical results - Fixed teams

- We expect the episode reward of ParetoPlayer to be higher than the episode reward of Player ($p \leq 2.2 \cdot 10^{-16}$).
- Training runs of ParetoPlayer tends to produce higher reward values ($p \leq 2.886 \cdot 10^{-12}$), but in some cases the rewards are almost equivalent.
- During evaluation ParetoPlayer tends to win more ($p \leq 3.048 \cdot 10^{-5}$)

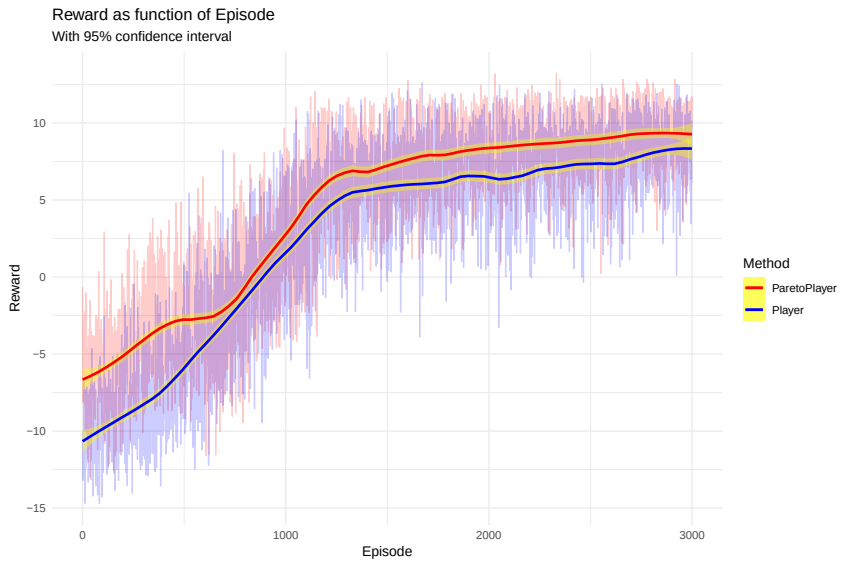

Figure 7: Row-mean reward per episode for Pareto and ParetoPlayer

Empirical results - Fixed Teams

Empirical results - Sampled teams

- We still expect the episode reward of ParetoPlayer to be higher than the episode reward of Player
- ParetoPlayers' reward distributions have a significant shift location to the right w.r.t to the Player's distribution ($p \leq 0.002278$ and $p \leq 0.01931$)
- The winning percentage is not always in favour of ParetoPlayer (0.716 and 0.673 vs 0.694)

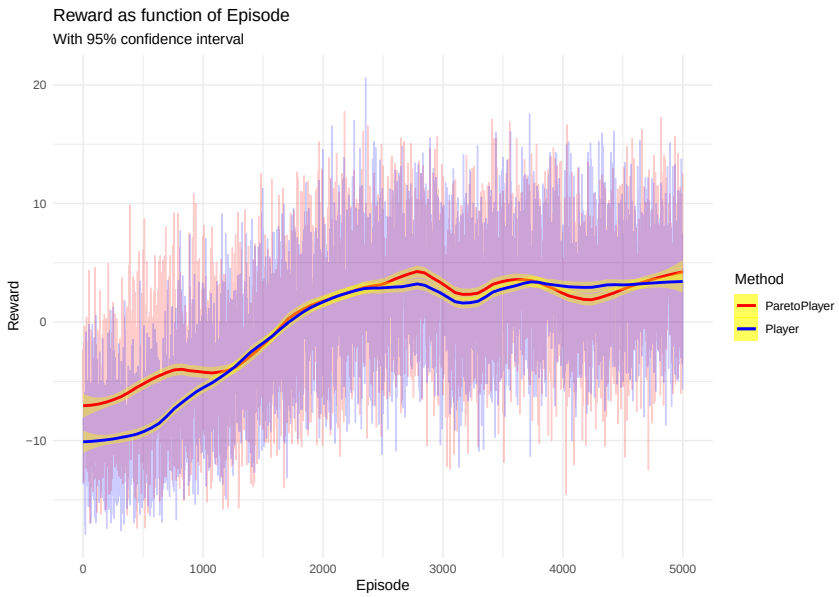

Figure 8: Row-mean reward per episode for Pareto and ParetoPlayer (sampled teams)

Empirical results - Sampled teams [First Game]

Empirical results - Sampled teams [Second Game]

Difficulties

The main difficulties we have encountered concern:

- Damage calculator
- Hyperparameters selection and topology search
- Pokémon double battles
- Pokémon battle switches

UNIVERSITÀ DI TRENTO

Conclusions

- ParetoPlayer is able to positively bias the training by providing higher rewards
- when the search space is small enough and a single win condition is presented, Player outperforms ParetoPlayer

Future works

- perform better topology and hyperparameters search
- reduce NSGA-II performance bottleneck (time-consuming operations)
- use another network to properly address forced switch

UNIVERSITÀ DI TRENTO

Thanks for your attention!

UNIVERSITÀ
DI TRENTO

Resources

Repositories

- pareto-epsilon-greedy-RL
- poke-env (modified)
- Pokemon_info

Collaborators' Github

- Simone Alghisi
- Samuele Bortolotti
- Massimo Rizzoli
- Erich Robbi

UNIVERSITÀ DI TRENTO

Normality - Fixed teams

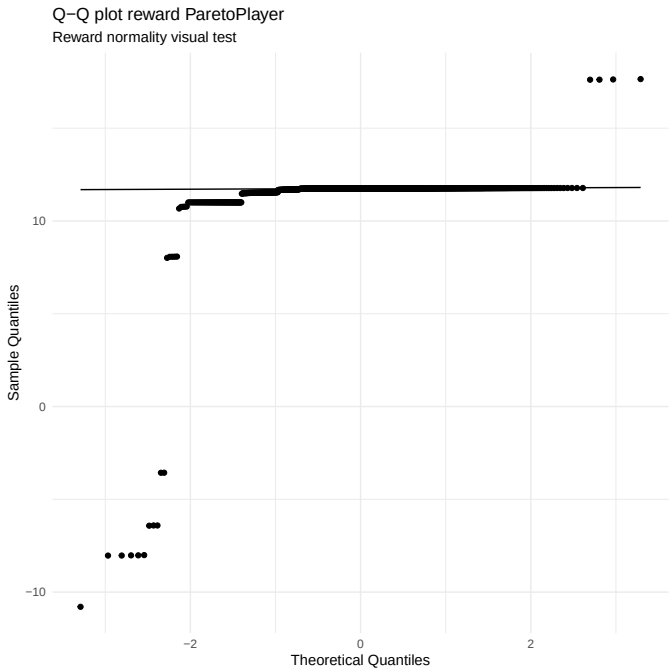

Figure 9: Quantile-Quantile plot episode reward computed on 1000 battles during ParetoPlayer model evaluation

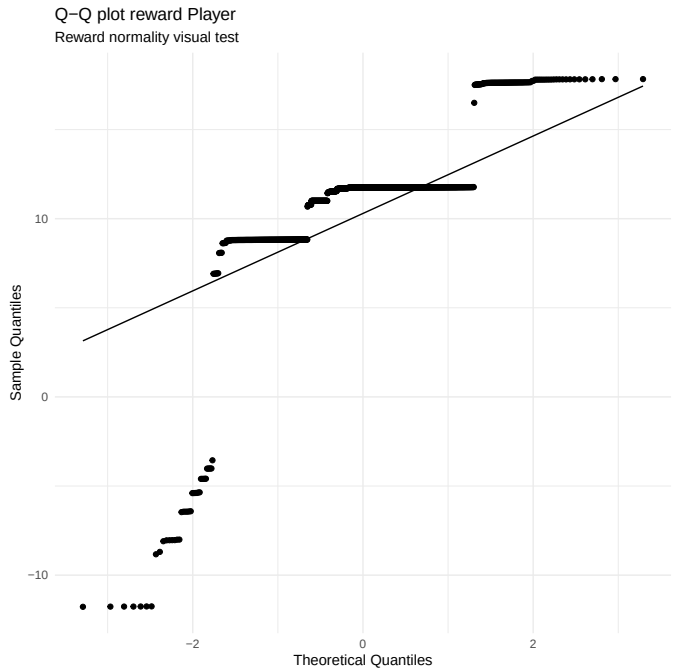

Figure 10: Quantile-Quantile plot episode reward computed on 1000 battles during Player model evaluation

UNIVERSITÀ DI TRENTO

Additional results - Box plots

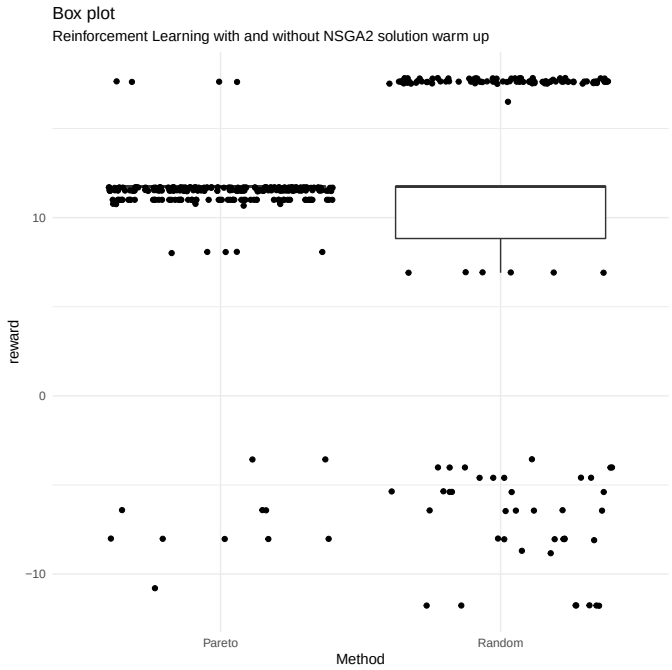

Figure 11: Box plot computed on 1000 battles during ParetoPlayer and Player model evaluation

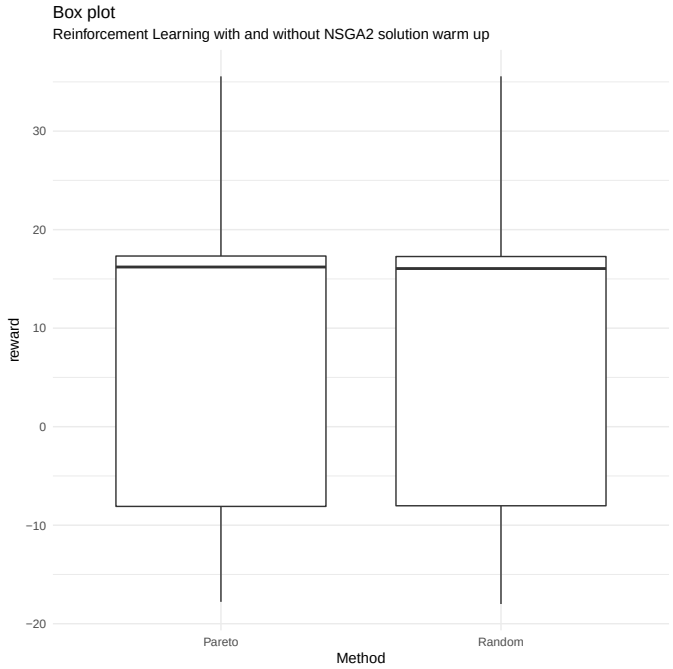

Figure 12: Box plot computed on 1000 battles during ParetoPlayer and Player model evaluation (with variable enemy team)

UNIVERSITÀ DI TRENTO